	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	108/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía Práctica de Estudio 9


Árboles parte 2

Elaborado por:

M.I. Elba Karen Sáenz García

Revisión:

Ing. Laura Sandoval Montaña

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	109/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía Práctica 9

Estructura de datos y Algoritmos II

Árboles. Parte 2.

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

Actividades

Implementar algunas operaciones realizadas sobre un árbol-B en algún lenguaje de programación.

Antecedentes

- Análisis previo del concepto de árbol-B y su representación visto en clase teórica.
- Manejo de listas, diccionarios, estructuras de control, funciones y clases en Python 3.
- Conocimientos básicos de la programación orientada a objetos.


Introducción

En los sistemas computacionales se cuenta con dos sistemas de almacenamiento en dos niveles, el almacenamiento en memoria principal y el almacenamiento secundario (como en discos magnéticos u otros periféricos).

En los almacenamientos en memoria secundaria el costo es más significativo y viene dado por el acceso a este tipo de memoria, muy superior al tiempo necesario para acceder a cualquier posición de memoria principal. Entonces al tratarse de un dispositivo periférico y, por tanto, de acceso lento, resulta primordial minimizar en lo posible el número de accesos.

Para realizar un acceso a un disco (para lectura / escritura) se requiere de todo un proceso de movimientos mecánicos que toman un determinado tiempo, por lo que para amortizar el tiempo gastado en esperar por todos esos movimientos la información es dividida en páginas del mismo tamaño en bits que se encuentran en las llamadas pistas del disco, así cada lectura y/o escritura al disco es de una o más páginas.

Los **árboles-B** (en inglés B-Tree), se utilizan cuando la cantidad de datos que se está trabajando es demasiado grande que no cabe toda en memoria principal. Así que, son árboles de búsqueda balanceados diseñados para trabajar sobre discos magnéticos u otros accesos o dispositivos de almacenamiento secundario.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	110/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

En los algoritmos que utilizan árboles-B se copian las páginas necesarias desde el disco a memoria principal y una vez modificadas las escriben de regreso al disco.

El tiempo de ejecución del algoritmo que utilice un árbol-B depende principalmente del número de lecturas y escrituras al disco, por lo que se requiere que estas operaciones lean y escriban lo más que se pueda de información y para ello cada nodo de un árbol-B es tan grande como el tamaño de una página completa lo que pone un límite al número de hijos de cada nodo.

Para un gran árbol-B almacenado en disco, se tienen factores de ramificación de entre 50 y 2000, dependiendo del número de llaves o valores que tenga cada nodo (tamaño de la página). Un factor grande de ramificación reduce tanto el peso del árbol como el número de accesos al disco requerido para encontrar una llave o valor.

En la figura 9.1 se muestra un árbol-B con un factor de ramificación de 1001 y peso 2 que puede almacenar más de un billón de llaves. Dentro de cada nodo x se encuentran el atributo $x.n$ que indica el número de llaves que son 1000 por nodo.

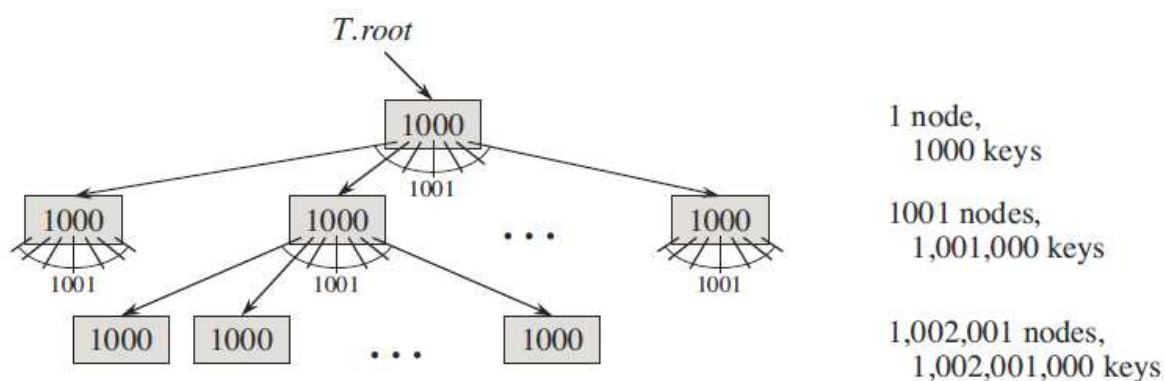



Figura 9.1. [1]

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	111/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Árboles B

Un Árbol-B es un árbol multirrama (con raíz T.Raiz) y tiene las siguientes propiedades:


- 1) Cada nodo x tiene la forma

$$((x.h_1, x.llave_1), (x.h_2, x.llave_2), (x.h_3, x.llave_3), \dots, (x.h_n, x.llave_n))$$

lo que indica que como mucho tiene n hijos $(x.h_1, x.h_2, \dots, x.h_n)$ y cuenta con las siguientes propiedades.

- Tiene $x.n$ llaves [o pares (llave, valor)] almacenadas en el nodo x .
 - Las $x.n$ llaves, $x.llave_1, x.llave_2, x.llave_3, \dots, x.llave_n$ están ordenadas y almacenadas en orden creciente tal que $x.llave_1 \leq x.llave_2 \leq x.llave_3 \leq \dots \leq x.llave_n$.
 - Puede ser hoja o nodo interno, se utiliza el atributo $x.hoja$ que contiene un valor verdadero si es hoja y falso si es nodo interno.
- 2) Cada nodo interno contiene $x.n + 1$ referencias o apuntadores a sus hijos $x.h_1, x.h_2, x.h_3, \dots, x.h_{n+1}$. Los nodos hojas no tienen hijos.
- 3) Las llaves $x.llave_i$ separan los rangos de las llaves almacenadas en cada sub-árbol. Si k_i es una llave almacenada en el sub-árbol $x.h_i$ entonces:
- $$k_i \leq x.llave_1 \leq x.llave_2 \leq x.llave_3 \leq \dots \leq x.llave_n \leq x.llave_{n+1}$$
- 4) Todas las hojas tienen la misma profundidad y el árbol tiene profundidad h .
- 5) Existe una cota superior e inferior sobre el número de llaves que puede contener un nodo. Esta cota puede ser expresada en términos de un entero $t \geq 2$ llamado el grado mínimo del árbol-B:
- Todo nodo que no sea la raíz debe tener al menos $t - 1$ llaves. Todo nodo interno que no sea la raíz debe tener al menos t hijos. Si el árbol es vacío, la raíz debe tener al menos una llave.
 - Todo nodo puede contener a lo más $2t - 1$ llaves. Por lo tanto, un nodo interno puede tener a lo más $2t$ hijos. Se dice que el nodo está lleno si este contiene exactamente $2t - 1$ llaves. El árbol-B más simple ocurre para cuando $t = 2$. Todo nodo interno entonces tiene ya sea 2, 3, o 4 hijos, también llamado árbol 2-3-4.

En la figura 9.2 se muestra el ejemplo de un árbol de altura 3 donde su grado mínimo es $t = 2$ y cada nodo puede tener a lo más $2t - 1$ llaves, en este caso 3 y al menos $t - 1$, que es 1.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	112/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

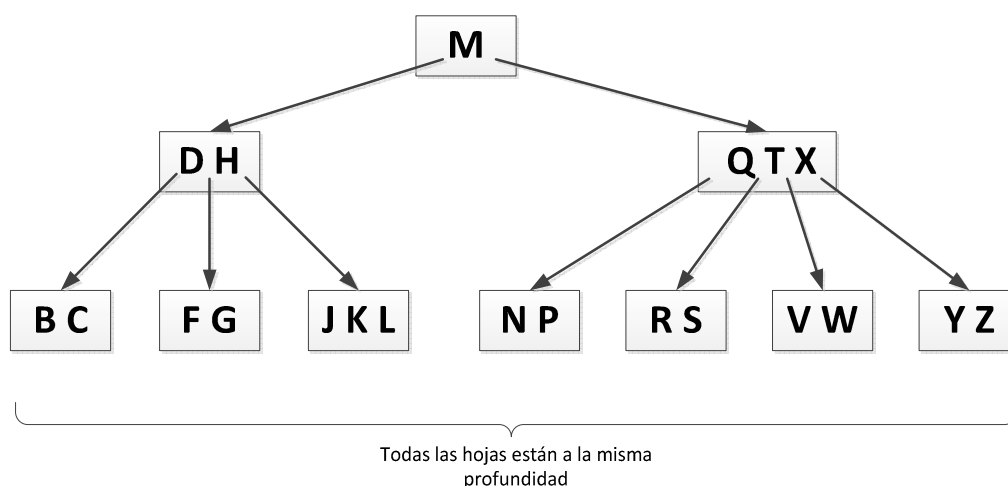


Figura 9.2 [1]

Operaciones Básicas

Algunas de las operaciones básicas realizadas en árboles B son búsqueda, inserción y eliminación de una llave. En esta práctica se trabajará con la búsqueda y la inserción.

Creación del árbol

Para la creación del árbol, lo primero es crear un nodo vacío y después ir insertando llaves e ir creando otros nodos si es necesarios. Para ello en este documento se utilizan los procedimientos propuestos en [1] B-TREE-CREATE() para crear un nodo raíz vacío y B-TREE-INSERT() para agregar nuevas llaves. En ambas funciones se utiliza un procedimiento auxiliar ALLOCATE-NODE(), el cual asigna una página del disco a un nuevo nodo. Se asume que cada nodo creado no requiere una lectura a disco desde que todavía no se utiliza la información almacenada en el disco para ese nodo.

El pseudocódigo del procedimiento B-TREE-CREATE () es [1]:

B-TREE-CREATE (T)


Inicio

```

x=ALLOCATE-NODE()
x.hoja=Verdadero
x.n=0
escribirDisco(x)
T.raiz=x

```

Fin

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	113/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Insertar una llave

La inserción es un poco más complicada que en un árbol binario. En un árbol-B no se puede solo crear una nueva hoja e insertarla porque daría lugar a ya no tener un árbol-B. Lo que se hace es insertar una nueva llave en un nodo hoja existente. Como no se puede insertar una llave en un nodo hoja que está lleno, se introduce una operación que divide el nodo lleno (que tiene $2t - 1$ llaves) en dos, en torno a la llave del medio ($y.llave_i$). Los dos nodos van a tener $t - 1$ llaves cada uno, y la llave del medio ($y.llave_i$) se moverá a su nodo padre para identificar el punto de división entre los dos nuevos árboles (Figura 9.3). Pero si el padre está lleno, también se debe dividir antes de que se inserte la nueva llave, por lo que el proceso se repetiría con los nodos más arriba hasta llegar al nodo raíz, en ese caso se genera un nuevo nodo raíz que contendrá solo el elemento desplazado hacia arriba de la antigua raíz.

De esta forma, un árbol-B crece por la raíz, de manera que, siempre que la altura se incrementa en 1, la nueva raíz sólo tiene un elemento.

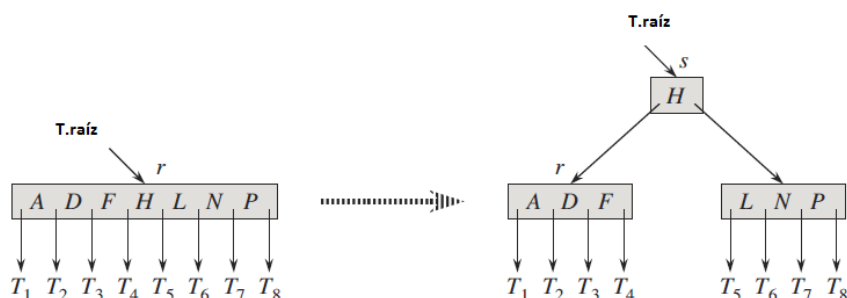



Figura 9.3 [1]

Para insertar en una sola pasada del nodo raíz a la hoja, primero se recorre el árbol buscando la posición donde se colocará la llave y en el camino se revisan qué nodos están llenos para dividirlos antes de realizar la inserción.

Cuando la llave encuentra su lugar en un nodo que no está lleno el proceso de inserción queda limitado a dicho nodo, pero si no lo está, el proceso puede implicar la creación de un nuevo nodo.

Ejemplo:

En la figura 9.4 se presenta un árbol-B donde su grado mínimo es $t = 3$, tal que cada nodo puede mantener 5 llaves.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	114/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

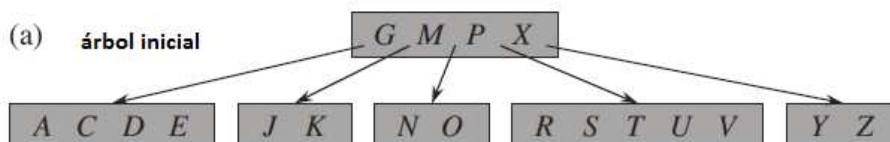


Figura 9.4. [1]

En las siguientes imágenes con color gris claro se muestran los nodos que se modifican conforme se van insertando las llaves que se mencionan.

Para insertar la *B*, se localiza primero donde se colocará la llave (en el nodo *ACDE*), y como tanto el nodo donde se insertará como su raíz no está lleno, solo se coloca en la posición adecuada. Figura 9.5.

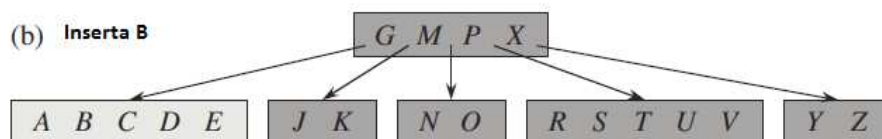


Figura 9.5. [1]

Ahora para insertar *Q* el nodo *RSTUV* se divide en 2 nodos que contienen *RS* y *UV*. La llave *T* se mueve a la raíz y *Q* se inserta al inicio de la mitad de la izquierda (nodo *RS*). Figura 9.6

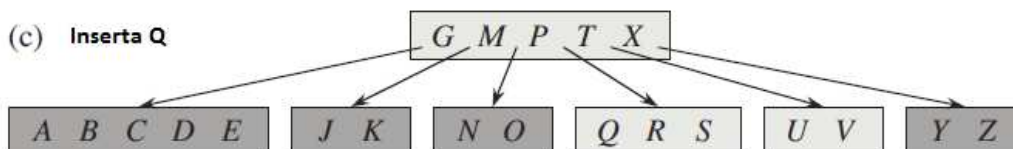



Figura 9.6. [1]

A partir del árbol anterior, para insertar la llave *L*, como la raíz está llena se divide en dos, además de formarse un nuevo nodo que es la nueva raíz. Después *L* se inserta en el nodo hoja que contiene *JK*. Figura 9.7.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	115/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

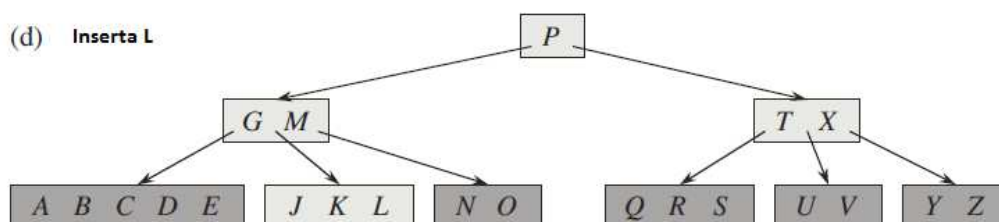


Figura 9.7 [1]

Finalmente, para insertar F , el nodo $ABCDE$ se divide en dos y el valor de en medio, en este caso C se coloca en el nodo padre, que no está lleno. Figura 9.8.

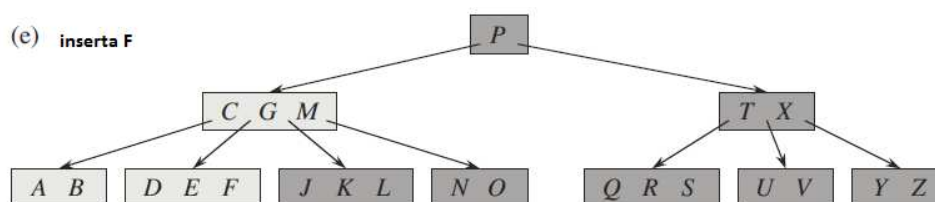


Figura 9.8 [1]

Un pseudocódigo de la inserción es el siguiente:

B-TREE-INSERT(T, k)

Inicio

$r = T.raiz$

Si $r.n == 2t - 1$

$s = \text{ALLOCATE-NODO}()$

$T.raiz = s$

$s.hoja = \text{FALSO}$

$s.n = 0$

$s.h_1 = r$

B-TREE-SPLIT-CHILD($s, 1$)


B-TREE-INSERT-NONFULL(s, k)

En otro caso

B-TREE-INSERT-NONFULL(r, k)

Fin Si

Fin

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	116/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Este procedimiento tiene dos casos, primero cuando el nodo raíz está lleno ($r.n == 2t - 1$), éste se divide (con B-TREE-SPLIT-CHILD()) y se forma un nuevo nodo s que será la nueva raíz, esto hace que se incremente la altura del árbol. El segundo caso se da cuando la raíz no está llena y se puede insertar la llave con TREE-INSERT-NONFULL().

El procedimiento B-TREE-INSERT-NONFULL() es recursivo, de forma tal que asegura revisar los niveles de abajo del árbol para insertar la llave en el nodo y posición adecuada y también garantiza que el nodo que se analiza, si está lleno, sea dividido con el llamado a la función B-TREE-SPLIT-CHILD() cuando sea necesario. A continuación, se muestran los pseudocódigos de estas funciones [1].

B-TREE-INSERT-NONFULL(x,k)

Inicio

$i = x.n$

Si $x.hoja == \text{verdadero}$

Mientras $i \geq 1$ y $k < x.llave_i$

$x.llave_{i+1} = x.llave_i$

$i = i - 1$

Fin Mientras

$x.llave_{i+1} = k$

$x.n = x.n + 1$

EscribirADisco(x)

En otro caso

Mientras $i \geq 1$ y $k < x.llave_i$

$i = i - 1$

Fin Mientras

$i = i + 1$

Leer Del Disco($x.h_i$)

Si $x.h_i.x == 2t - 1$

B-TREE-SPLIT-CHILD($x.i$)

Si $k > x.llave_i$

$i = i + 1$


Fin Si

Fin Si

B-TREE-INSERT-NONFULL($x.h_i, k$)

Fin Si

Fin

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	117/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

B-TREE-SPLIT-CHILD(x, i)

Inicio

$z = \text{ALLOCATE-NODE}()$

$y = x.h_i$

$z.hoja = y.hoja$

$z.n = t - 1$

Para $j = 1$ hasta $j = t - 1$

$z.llave_j = y.llave_{j+t}$

Fin Para

Si $y.hoja == \text{FALSO}$

Para $j = 1$ hasta t

$z.h_j = y.h_{j+t}$

Fin para

Fin Si

$y.n = t - 1$

Para $j = x.n + 1$ decrementando hasta $j = i + 1$

$x.h_{j+1} = x.h_j$

Fin Para

$x.h_{i+1} = z$

Para $j = x.n$ decrementando hasta $j = i$

$x.llave_{j+1} = x.llave_j$

Fin Para

$x.llave_i = y.llave_t$

$x.n = x.n + 1$

EscribirADisco(y)


EscribirADisco(z)

EscribirADisco(x)

Fin

Búsqueda

La búsqueda en un árbol-B se parece mucho a la búsqueda en un árbol binario, excepto que en lugar de hacerlo binario o de dos caminos, se hace una decisión de múltiples caminos de acuerdo al número de hijos que tiene el nodo.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	118/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Un algoritmo general para buscar la llave k puede ser:

- 1-Seleccionar nodo actual igual a la raíz del árbol.
 - 2-Comprobar si la clave se encuentra en el nodo actual:
 - Si la clave está, termina algoritmo
 - Si la clave no está:
 - Si nodo actual es hoja, termina algoritmo
 - Si nodo actual no es hoja y n es el número de claves en el nodo,
 - Nodo actual == hijo más a la izquierda, si $k < k_1$
 - Nodo actual == hijo más a la derecha a la derecha, si $k > k_n$
 - Nodo actual == i -ésimo hijo, si $k_i < k < k_{i+1}$
- Volver al paso 2.

La figura 9.9 se puede observar un árbol-B cuyas llaves son las letras del alfabeto, cada nodo x contiene $x.n$ llaves y tiene $x.n + 1$ hijos, y además todas las hojas tienen la misma profundidad. También se ilustra en gris claro el progreso de la búsqueda de la llave R en ese árbol.

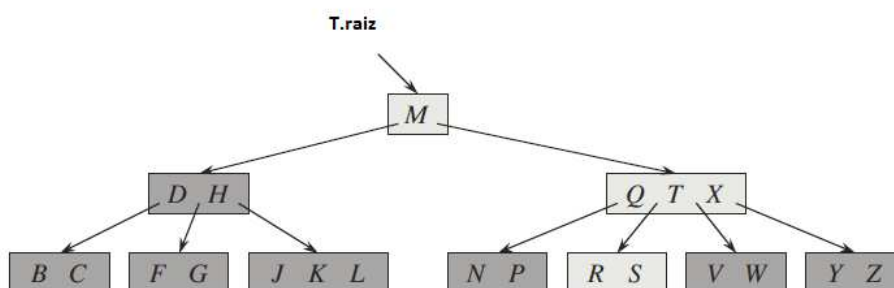



Figura 9.9[1]

Una manera de plantear el algoritmo es de forma recursiva, a continuación se presenta un procedimiento para la búsqueda en árboles-B (propuesto en [1]) donde se toma como entrada un apuntador al nodo raíz x de cada sub árbol y una llave k que se buscará dentro de cada subárbol. La llamada inicial al procedimiento se realiza con el nodo raíz y la llave a buscar k , esto es, $BTreeSearch(T.raiz, K)$. Si la llave k está dentro del árbol-B la función retorna el par ordenado (y, i) consistente de un nodo y y un índice i tal que $y.llave_i = k$. De otra forma el procedimiento regresa nulo.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	119/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

B-TREE-SEARCH(x, k)

Inicio

$i=1$

Mientras $i \leq x.n$ y $k > x.llave_i$

$i=i+1$

Fin Mientras

Si $i \leq x.n$ y $k > x.llave_i$

retorna (x, i)

Si no

Si $x.hoja == verdadero$

Retorna Ninguno

Si no

Leer-Disco($x.h_i$)

Retorna B-TREE-SEARCH($x.h_i, k$)

Fin Si

Fin Si

Fin


En este procedimiento se utiliza inicialmente una búsqueda lineal para encontrar el índice i tal que $k \leq x.llave_i$ o de lo contrario se coloca i a $x.n + 1$. Si se ha descubierto la llave k , se retorna el nodo x junto con su ubicación (x, i) . De otra forma se determina que la búsqueda no fue exitosa porque el nodo x es una hoja, o se llama recursivamente a la función B-TREE-SEARCH() indicando el subárbol de x adecuado, para seguir buscando. Los parámetros de la nueva llamada a B-TREE-SEARCH($x.h_i, k$) se obtiene después de efectuar una lectura a disco para obtener la información de ese nodo hijo.

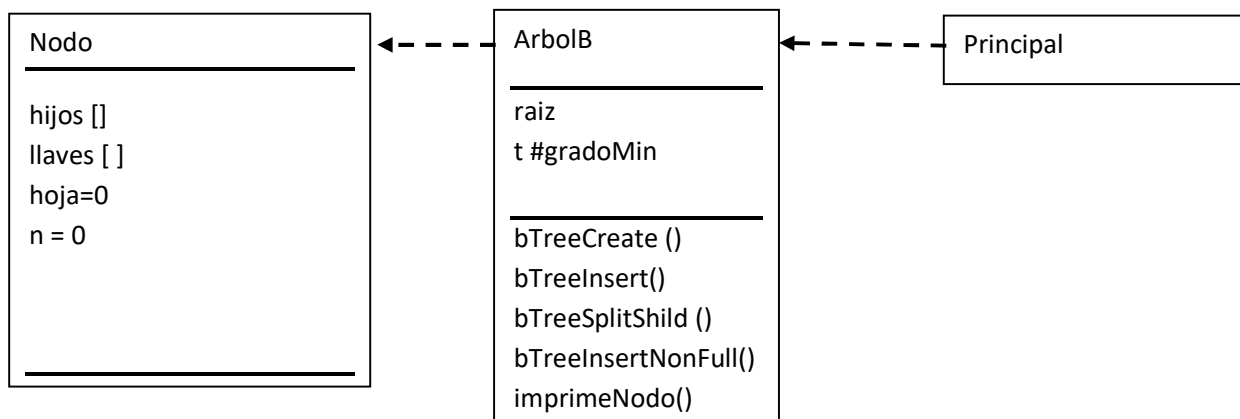
Desarrollo

Actividad 1

Se realizará de forma guiada un programa en Python que permita crear un árbol B e ir insertando datos. Para ello se utilizarán los pseudocódigos de los procedimientos explicados en este documento.

Primero se plantea el siguiente diagrama de clases:

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	120/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			



Para la realización del programa se asumirá que los datos de los nodos ya no están en disco. Es decir, todo se trabajará en memoria principal.

La clase Nodo queda:

```

#Autor Elba Karen Sáenz García


class Nodo:
    def __init__(self,t):
        self.hijos = list()
        self.llaves = list()
        self.hoja=1
        self.n=0
        for k in range(2*t):
            self.llaves.append([None])
        for k in range(2*t+1):
            self.hijos.append([None])
  
```

El inicio de la clase ArbolB es:

```

#Auto Elba Karen Sáenz García

class ArbolB:
    def __init__(self,gradoMinimo):
        self.t= gradoMinimo
        self.raiz = None
  
```

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	121/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Ahora se muestran los métodos que hay que ir agregando a la clase ArbolB de acuerdo al diagrama de clases planteado y los pseudocódigos vistos en este documento.

El método bTreeCreate() crea un nuevo nodo :

```
def bTreeCreate(self):
    if(self.raiz == None):
        self.raiz = Nodo(self.t)
    return self.raiz
```

El método bTreeSplitShild():


```
def bTreeSplitShild(self,x,i):
    z=Nodo(self.t)
    y = x.hijos[i]
    z.hoja=y.hoja
    z.n=self.t-1

    for j in range(1,self.t):
        z.llaves[j]=y.llaves[j+self.t]
        y.llaves[j+self.t]=None

    if y.hoja==0:
        for j in range(1,self.t+1):
            z.hijos[j]=y.hijos[j+self.t]
            y.hijos[j+self.t]=None
    y.n=self.t-1
    for j in range(x.n+1,i,-1):
        x.hijos[j+1]=x.hijos[j]

    x.hijos[i+1]=z

    for j in range (x.n,i-1,-1):
        x.llaves[j+1]=x.llaves[j]
    x.llaves[i]=y.llaves[self.t]
    y.llaves[self.t]=None
    x.n=x.n+1
```


	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	122/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

El método bTreeInsertNonFull():

```
def bTreeInsertNonFull(self, x, k):
    i=x.n
    if x.hoja == 1:
        while ( i >= 1) and (k < x.llaves[i]):
            x.llaves[i+1]= x.llaves[i]
            i=i-1
        x.llaves[i+1]=k
        x.n=x.n+1
        #escribir a disco
    else:
        #No es hoja
        while (i >= 1) and (k < x.llaves[i]):
            i=i-1
        i=i+1
        #leer disco
        if x.hijos[i].n == 2*self.t-1:
            self.bTreeSplitShild(x,i)
            if k > x.llaves[i]:
                i=i+1
            self.bTreeInsertNonFull(x.hijos[i],k)
```

El método bTreeInsert():

```
def bTreeInsert(self,nodo, k):
    r=self.raiz
    #nodo lleno
    if r.n == 2*self.t-1:
        s=Nodo(self.t)
        self.raiz=s
        s.hoja=0
        s.n=0
        s.hijos[1]=r
        self.bTreeSplitShild(s,1)
        self.bTreeInsertNonFull(s,k)
    else:
        self.bTreeInsertNonFull(r,k)
```

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	123/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

El método `imprimeNodo()`:

```
def imprimeNodo(self,nodo):
    for i in range(1,2+self.t,1):
        if (nodo.llaves[i] != None):
            print( nodo.llaves[i])
```

Una vez terminadas las dos clases se requiere realizar una controladora que permita ir formando un árbol B, creándolo e insertando la siguiente secuencia.

B,T,H,M,O,C,Z,G,L,E,N,P,R,D,J,Q,F,W,X.

Además, se irá mostrando la formación del árbol.

En la controladora, primero se colocará la inserción de B, T y H como se muestra en el siguiente código:

```
BT=ArbolB(2)

actual=BT.bTreeCreate()


print ("Se insertara B")
#print (BT.raiz.llaves)
BT.bTreeInsert(actual,ord("B"))

print ("Se insertara T")
BT.bTreeInsert(actual,ord("T"))

print ("Se insertara H")
BT.bTreeInsert(actual,ord("H"))

print ("Imprime raíz")
BT.imprimeNodo(BT.raiz)
```

. El proceso que se realiza se observa en la figura 9.10.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	124/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

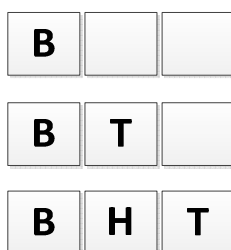


Figura 9.10

Después se insertará M y como ya no hay lugar en el nodo, se realiza la división de mismo y la creación de uno nuevo, como se muestra en el siguiente código y en la figura 9.11:

```
print ("Se insertara M")
BT.bTreeInsert (actual,ord("M"))

print (BT.raiz.llaves)
print (BT.raiz.hijos[1].llaves)
print (BT.raiz.hijos[2].llaves)
```

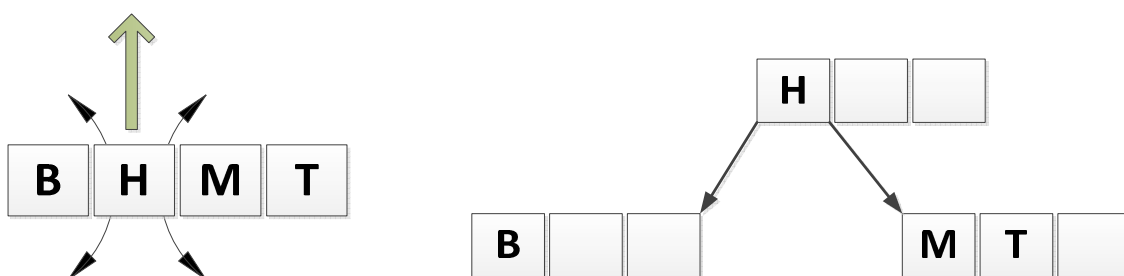



Figura 9.11

Ahora se insertarán las llaves O y C, además de visualizar la raíz y sus hijos. Como hay espacio en los nodos solo se colocan en el lugar correspondientes. El árbol resultante se muestra en la figura 9.12.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	125/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```

print ("Se insertara O")
BT.bTreeInsert(actual,ord("O"))
print ("Se insertara C")

BT.bTreeInsert(actual,ord("C"))
print (BT.raiz.llaves)
print (BT.raiz.hijos[1].llaves)
print (BT.raiz.hijos[2].llaves)

```

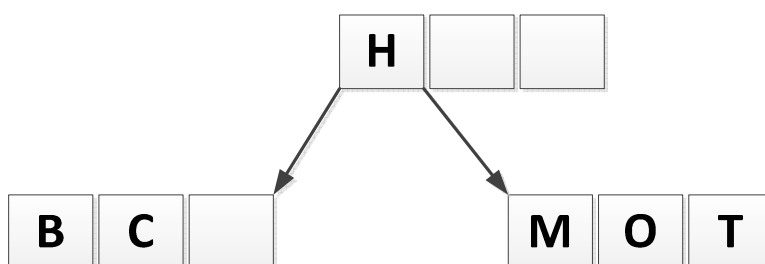


Figura 9.12


Cuando se inserta Z, no hay lugar en el nodo correspondiente, así que se realiza una división y creación de nodo.
Figura 9.13.

```

print ("Se insertara Z")
BT.bTreeInsert(actual,ord("Z"))

print (BT.raiz.llaves)
print (BT.raiz.hijos[1].llaves)
print (BT.raiz.hijos[2].llaves)
print (BT.raiz.hijos[3].llaves)

```

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	126/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

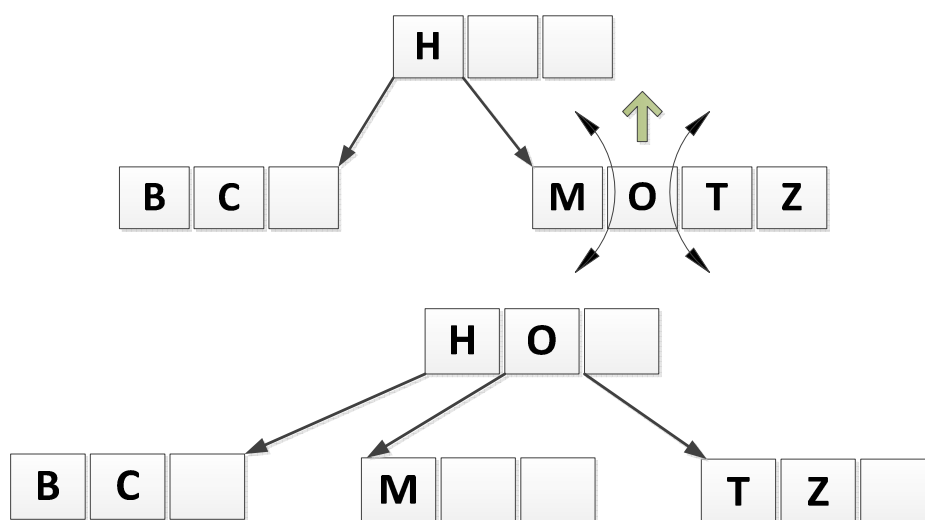


Figura 9.13

Actividad 2

Terminar de insertar la secuencia e ir mostrando en la salida estándar y dibujar como va quedando el árbol.

Actividad 3

Ejercicios sugeridos por el profesor

Referencias

[1]CORMEN, Thomas, LEISERSON, Charles,et al.

Introduction to Algorithms

3rd edition

MA, USA

The MIT Press, 2009

[2]Ziviani, Nivio

Diseño de algoritmos con implementaciones en Pascal y C

Ediciones paraninfo /Thomson Learning

2007

[3] <https://www.youtube.com/watch?v=HHoWd93mKjA>