

UNIVERSIDAD AUTONOMA METROPOLITANA

ALMACENAMIENTO Y ESTRUCTURA DE ARCHIVOS

Proyecto final
Árbol B

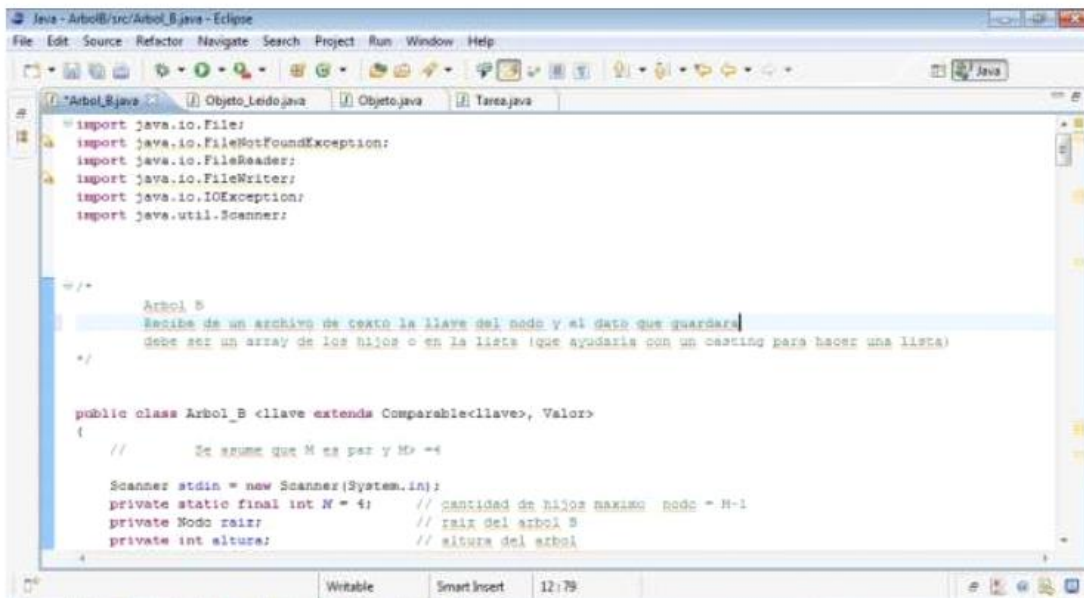
EQUIPO:

*Flores Ornelas Demart	210208733
*Vélez Suárez del Real Luis Gerardo	210201359
*Álvarez Posadas Victor Hugo	210303892

Introducción:

El Árbol B es una estructura de datos muy importante en ciencias de la computación. Son estructuras no lineales al contrario que los arrays que constituyen estructuras lineales. Son muy utilizados en informática para representar fórmulas algebraicas como un método eficiente para búsquedas grandes y complejas, listas dinámicas y aplicaciones diversas tales como inteligencia artificial o algoritmos de cifrado. Casi todos los sistemas operativos almacenan sus archivos en Árboles o estructuras similares a Árboles. Además, los árboles se utilizan en diseño de compiladores, proceso de texto y algoritmos de búsqueda. Nuestro árbol B que generamos deberá poder hacer inserción y recorrido en orden con registros leídos de un archivo de texto llamado números.txt.

DESARROLLO:



```
Java - Arbol_B/src/Arbol_B.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Arbol_B.java
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

/**
 * Arbol B
 * escribe de un archivo de texto la llave del nodo y el dato que guardara
 * debe ser un array de los hijos o en la lista (que ayudaria con un casting para hacer una lista)
 */

public class Arbol_B <llave extends Comparable<llave>, Valor>
{
    // Se assume que M es par y M% = 4

    Scanner stdin = new Scanner(System.in);
    private static final int N = 4; // cantidad de hijos maximo  nodo = M-1
    private Nodo raiz; // raiz del arbol B
    private int altura; // altura del arbol
}
```

```
Java - ArbolB/src/ArbolB.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

ArbolB.java ObjetoLeido.java Objeto.java Tarea.java

public class ArbolB <llave extends Comparable<llave>, Valor>
{
    // Se asume que N es par y M = N-1

    Scanner stdin = new Scanner(System.in);
    private static final int N = 4; // cantidad de hijos maximo nodo = M-1
    private Nodo raiz; // raiz del arbol B
    private int altura; // altura del arbol
    private int N; // numero de pares de llaves

    // ayudante del Arbol B de datos del tipo de nodo
    private static final class Nodo
    {
        private int m; // numero de hijos
        private Entrada[] hijos = new Entrada[M]; // Arreglo de hijos
        private Nodo(int k)
        {
            m = k; // Crea un nodo con k hijos
        }
    }

    // nodos internos: llave y siguiente
}
```

```
Java - ArbolB/src/ArbolB.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

ArbolB.java ObjetoLeido.java Objeto.java Tarea.java

private Valor Buscar(Nodo x, int i, int altura)
{
    Entrada[] Hijos = x.hijos;

    // Nodo externo
    if (altura == 0) // altura
    {
        for (int j = 0; j < x.m; j++)
        {
            if (iguales(i, Hijos[j].llave))
                return (Valor) Hijos[j].valor; // Compara con el metodo iguales
        }
    }

    // Nodo interno
    else {
        for (int j = 0; j < x.m; j++)
        {
            if (j+1 == x.m || menores(i, Hijos[j+1].llave))
                return Buscar(Hijos[j].siguiente, i, altura-1);
        }
    }
    return null;
}
```

```
Java - ArbolB/src/Arbol_B.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Arbol_B.java Objeto_Leido.java Objeto.java Tarea.java

// nodos internos: llave y siguiente
// nodos externos: llave y valor
private static class Entrada
{
    private Comparable llave;
    private Object valor;
    private Nodo siguiente; // ayudante para repetir entradas de matriz

    //Constructor con argumentos de la clase Entrada
    public Entrada(Comparable llave, Object valor, Nodo siguiente)
    {
        this.llave = llave;
        this.valor = valor;
        this.siguiente = siguiente;
    }

    // constructor sin argumentos inicializa el arbol (raiz)
    public Arbol_B()
    {
        raiz = new Nodo(0);
    }
}
```

```
Java - ArbolB/src/Arbol_B.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Arbol_B.java Objeto_Leido.java Objeto.java Tarea.java

}

// Regresa el número de pares clave-valor en el Arbol
public int tamaño()
{
    return N;
}

// Metodo que regresa la altura del Arbol
public int Altura()
{
    return altura;
}

// Búsqueda de la clave dada, el valor de retorno asociados; return null si no hay clave
public Valor get(int i)
{
    return Buscar(raiz, i, altura);
}

private Valor Buscar(Nodo n, int i, int altura)
{
}
```

```
Java - ArbolB/src/ArbolB.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

ArbolB.java | ObjetoLeido.java | Objeto.java | Tarea.java

// inserta llave-valor
// agrega el dato para comprobar si hay duplicados de las llaves
public void agregar(int i, Valor valor)
{
    Nodo u = Insertar(raiz, i, valor, altura);
    M++;
    if (u == null) return;

    // necesita separar la raiz
    Nodo t = new Nodo(2);

    t.hijos[0] = new Entrada(raiz.hijos[0].llave, null, raiz);
    t.hijos[1] = new Entrada(u.hijos[0].llave, null, u);
    raiz = t;
    altura++;
}

private Nodo Insertar(Nodo h, int i2, Valor valor, int ht)
{
    int j;

```

```
Java - ArbolB/src/ArbolB.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

ArbolB.java | ObjetoLeido.java | Objeto.java | Tarea.java

private Nodo Insertar(Nodo h, int i2, Valor valor, int ht)
{
    int j;
    Entrada t = new Entrada(i2, valor, null);
    // nodo externo
    if (ht == 0)
    {
        for (j = 0; j < h.M; j++)
        {
            if (menores(i2, h.hijos[j].llave)) break;
        }
    }
    // nodo interno
    else {
        for (j = 0; j < h.M; j++)
        {
            if ((j+1 == h.M) || menores(i2, h.hijos[j+1].llave))
            {
                Nodo u = Insertar(h.hijos[j+1].siguiente, i2, valor, ht-1);
                if (u == null)
                    return null;
                t.llave = u.hijos[0].llave;
                t.siguiente = u;
                break;
            }
        }
    }

```

```
Java - ArbolB/src/ArbolB.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

ArbolB.java Objeto_Leido.java Objeto.java Tarea.java

break;
}
}
for (int i = h.m/ 2 > 0; i--> 0; h.hijos[i] = h.hijos[i-1];
h.hijos[i] = t;
h.m++;
if (h.m < M)
return null;
else
return Divide(h);
}

//Metodo divide el nodo
private Node Divide(Node h)
{
Node t = new Node(M/2);
h.m = M/2;
for (int j = 0; j < M/2; j++)
{
t.hijos[j] = h.hijos[M/2+j];
}
return t; //dentro ret o sin corchetes
}

Writable SmartInsert 180:5
```

```
Java - ArbolB/src/ArbolB.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

ArbolB.java Objeto_Leido.java Objeto.java Tarea.java

// Metodo que lee y regresa una cadena de caracteres
public String toString()
{
return toString(raiz, altura, "") + "\n";
}
private String toString(Node h, int ht, String indent)
{
String s = "";
Entrada[] hijos = h.hijos;

if (ht == 0)
{
for (int j = 0; j < h.m; j++)
{
s += indent + hijos[j].llave + " " + hijos[j].valor + "\n";
}
}
else
{
for (int j = 0; j < h.m; j++)
{
if (j > 0) s += indent + "(" + hijos[j].llave + ")\n";
s += toString(hijos[j].siguiente, ht-1, indent + " ");
}
}
}

Writable SmartInsert 180:5
```

```
Java - ArbolB/src/Arbol_B.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Arbol_B.java | Objeto_Leido.java | Objeto.java | Tarea.java

}
else
{
    for (int j = 0; j < h.mz[j]++)
    {
        if (j > 0) s += indent + "(" + hijos[j].llave + ")\n";
        s += toString(hijos[j].siguiente, h-1, indent + " ");
    }
    return s;
}

// Metodo que compara los nodos
private boolean menores(Comparable k1, Comparable k2)
{
    return k1.compareTo(k2) < 0;
}

private boolean iguales(Comparable k1, Comparable k2) {
    return k1.compareTo(k2) == 0;
}

Writable SmartInsert 160:5
```

```
Java - ArbolB/src/Arbol_B.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Arbol_B.java | Objeto_Leido.java | Objeto.java | Tarea.java

/* throws IOException
.....*/
public static void main(String[] args) throws IOException {
    Arbol_B<String, String> arbol = new Arbol_B<String, String>();

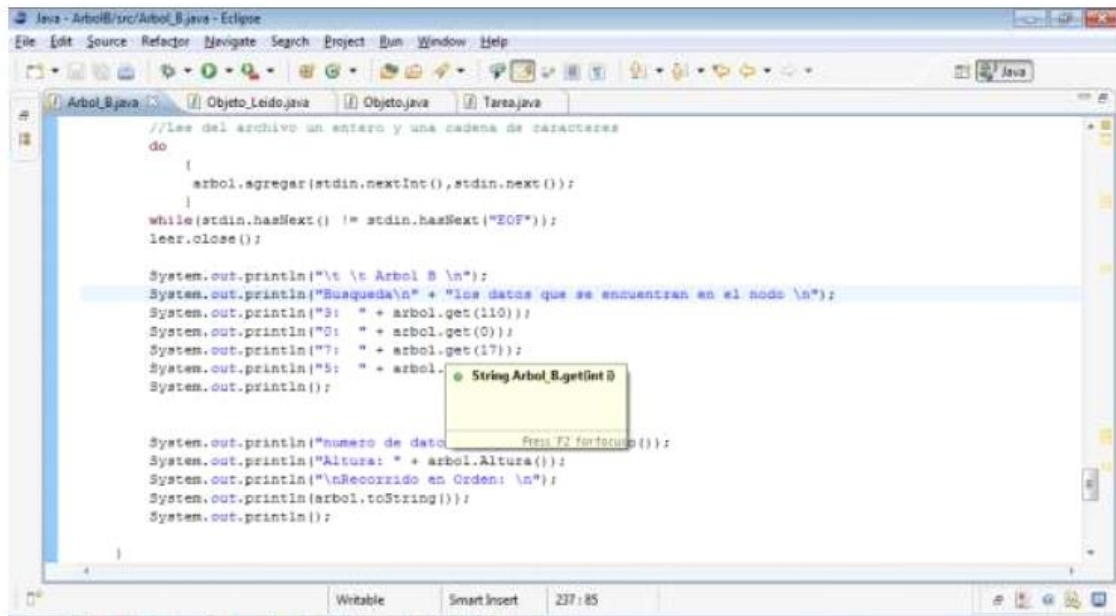
    Scanner stdin = new Scanner(System.in);
    stdin = new Scanner(new File("C:/tst/numeros.txt"));

    File abriz = new File("C:/tst/numeros.txt");
    FileReader leer = new FileReader(abriz);
    stdin = new Scanner(new File("C:/tst/numeros.txt"));

    int y=0;

    //Lee del archivo un entero y una cadena de caracteres
    do
    {
        arbol.agregar(stdin.nextInt(), stdin.next());
    }
    while(stdin.hasNext() != stdin.hasNext("EOF"));
    leer.close();
}

Writable SmartInsert 234:22
```

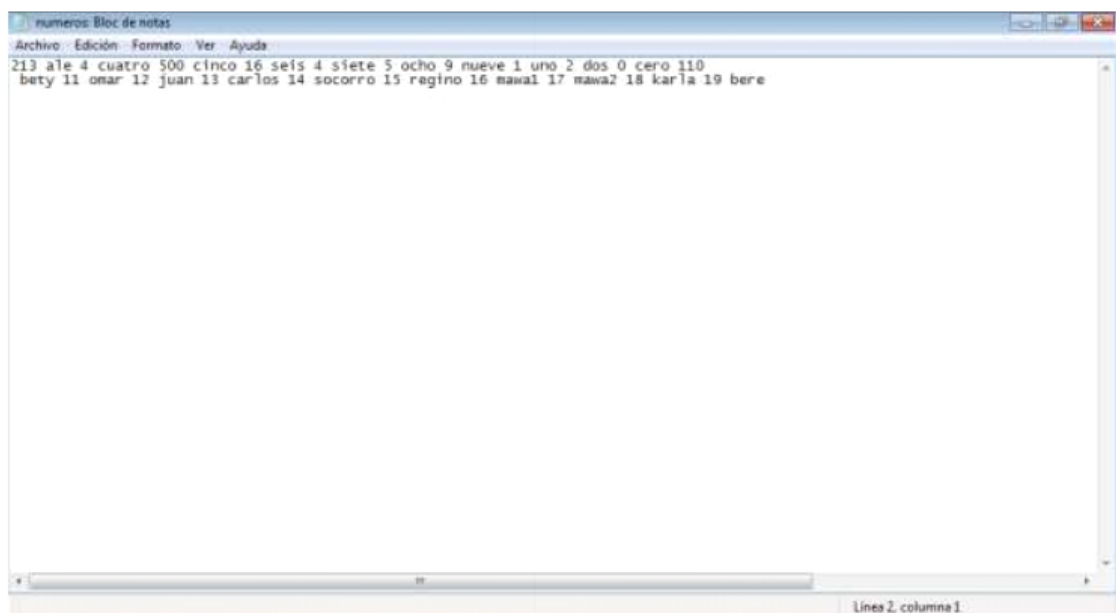
```
//lee del archivo un entero y una cadena de caracteres
do
{
    arbol.agregar(stdin.nextInt(),stdin.next());
}
while(stdin.hasNext() != stdin.hasNext("EOF"));
leer.close();

System.out.println("\t \t Arbol B \n");
System.out.println("Busqueda\n" + "los datos que se encuentran en el nodo \n");
System.out.println("3: " + arbol.get(110));
System.out.println("0: " + arbol.get(0));
System.out.println("7: " + arbol.get(17));
System.out.println("5: " + arbol.get(5));
System.out.println();

System.out.println("numero de datos: " + arbol.getint());
System.out.println("Altura: " + arbol.Altura());
System.out.println("\nRecorrido en Orden: \n");
System.out.println(arbol.toString());
System.out.println();

}
```

Archivo de texto “números.txt” del cual se reciben los datos:



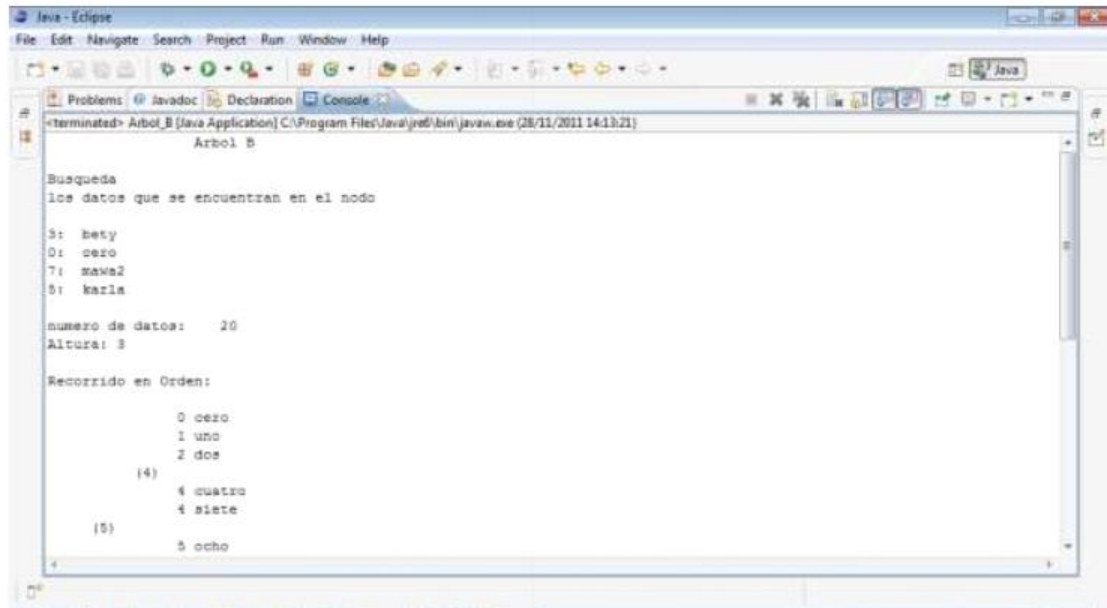
numeros: Bloc de notas

Archivo Edición Formato Ver Ayuda

213 ale 4 cuatro 500 cinco 16 seis 4 siete 5 ocho 9 nueve 1 uno 2 dos 0 cero 110
bety 11 onar 12 juan 13 carlos 14 socorro 15 regino 16 mawa1 17 mawa2 18 karla 19 bere

Línea 2, columna 1

EJECUCIÓN:



```
<terminated> Arbol_B [Java Application] C:\Program Files\Java\jre6\bin\java.exe (28/11/2011 14:13:21)
Arbol_B

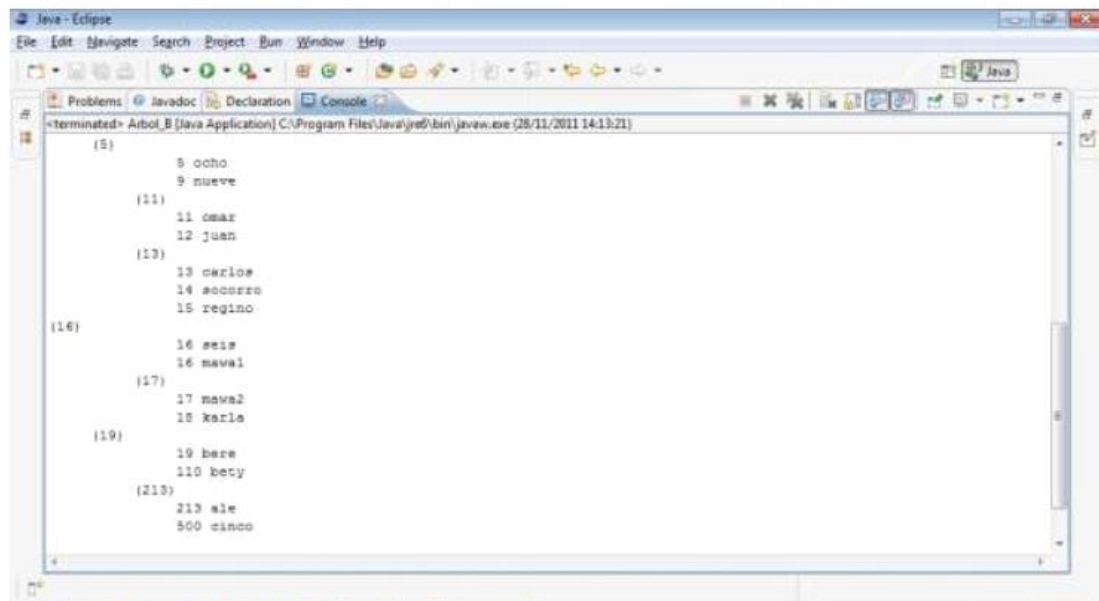
Busqueda
los datos que se encuentran en el nodo

3: bety
0: zero
7: maxa2
5: karla

numero de datos: 20
Altura: 3

Recorrido en Orden:

          0 zero
          1 uno
          2 dos
      (4)   4 cuatro
           4 siete
      (5)   5 ocho
```



```
      (5)   8 ocho
           9 nueve
      (11)  11 osar
           12 juan
      (13)  13 carlos
           14 socorro
           15 regino
      (16)  16 seis
           16 maval
      (17)  17 maxa2
           18 karla
      (19)  19 bere
           110 bety
      (213) 213 ale
           500 cinco
```

```

Codigo:
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

/*
Arbol B
Recibe de un archivo de texto la llave del nodo y el dato que guarda
debe ser un array de los hijos o en la lista (que ayudaría con un casting
para
hacer una lista)
*/

public class Arbol_B <llave extends Comparable<llave>, Valor>
{
    //Se asume que M es par y M> =4

    Scanner stdin = new Scanner(System.in);
    private static final int M = 4;    // cantidad de hijos máximo
nodo
    // nodo = M-1
    private Nodo raiz;                // raiz del arbol B
    private int altura;                // altura del arbol
    private int N;                    // numero de pares de llaves

    // ayudante del Árbol B de
    datos //del
    tipo de nodo

    private static final class Nodo
    {
        private int m;                // numero de hijos
        private Entrada[] hijos = new Entrada[M];    // Arreglo de hijos
        private Nodo(int k)
        {
            m = k;
        }
        //Crea un nodo con k hijos
    }

    // nodos internos:    llave y siguiente
    // nodos externos:    llave y valor

    private static class Entrada
    {
        private Comparable llave;
        private Object valor;
        private Nodo siguiente; //ayudante para repetir entradas de
        //matriz
    }
}

```

```

//Constructor con argumentos de la clase Entrada
    public Entrada(Comparable llave, Object valor, Nodo siguiente)
    {
        this.llave    = llave;
        this.valor    = valor;
        this.siguiente = siguiente;
    }
}
// constructor sin argumentos inicializa el arbol (raiz)

public Arbol_B()
{
    raiz = new Nodo(0);
}

// Regresa el numero de pares clave-valor en el Arbol

public int tamaño()
{
    return N;
}

// Metodo que regresa la altura del Arbol
public int Altura()
{
    return altura;
}

// búsqueda de la clave dada, el valor de retorno asociados; return
//null si no hay clave

public Valor get(int i)
{
    return Buscar(raiz, i, altura);
}

private Valor Buscar(Nodo x, int i, int altura)
{
    Entrada[] Hijos = x.hijos;

    // Nodo externo
    if (altura == 0) //altura
    {
        for (int j = 0; j < x.m; j++)
        {
            if (iguales(i, Hijos[j].llave))
                return (Valor) Hijos[j].valor;//Compara con el metodo iguales
        }
    }
}

```

```

// Nodo interno
else {
    for (int j = 0; j < x.m; j++)
    {
        if (j+1 == x.m || menores(i, Hijos[j+1].llave))
            return Buscar(Hijos[j].siguiente, i, altura-1);
    }
}
return null;
}

```

```

// inserta llave-valor
// agrega el dato para comprobar si hay duplicados de las llaves

```

```

public void agregar(int i, Valor valor)
{
    Nodo u = Insertar(raiz, i, valor, altura);
    N++;
    if (u == null) return;

    // necesita separar la raíz
    Nodo t = new Nodo(2);

    t.hijos[0] = new Entrada(raiz.hijos[0].llave, null, raiz);
    t.hijos[1] = new Entrada(u.hijos[0].llave, null, u);
    raiz = t;
    altura++;
}

private Nodo Insertar(Nodo h, int i2, Valor valor, int ht)
{ int j;
  Entrada t = new Entrada(i2, valor, null);
  // nodo externo
  if (ht == 0)
  {
      for (j = 0; j < h.m; j++)
      {
          if (menores(i2, h.hijos[j].llave)) break;
      }
  }
  // nodo interno
  else {
      for (j = 0; j < h.m; j++)
      {
          if ((j+1 == h.m) || menores(i2, h.hijos[j+1].llave))
          {
              Nodo u = Insertar(h.hijos[j+1].siguiente, i2, valor, ht-
1);

              if (u == null)
                  return null;
              t.llave = u.hijos[0].llave;
              t.siguiente = u;
              break;
          }
      }
  }
}

```

```

        }
    }
    for (int i = h.m; i > j; i--) h.hijos[i] = h.hijos[i-1];
    h.hijos[j] = t;
    h.m++;
    if (h.m < M)
        return null;
    else
        return Divide(h);
}

//Metodo divide el nodo

private Nodo Divide(Nodo h)
{
    Nodo t = new Nodo(M/2);
    h.m = M/2;
    for (int j = 0; j < M/2; j++)
    {
        t.hijos[j] = h.hijos[M/2+j];
    }
    return t;    //dentro ret o sin corchet
}

// Metodo que lee y regresa una cadena de caracteres

public String toString()
{
    return toString(raiz, altura, "") + "\n";
}

private String toString(Nodo h, int ht, String indent)
{
    String s = "";
    Entrada[] hijos = h.hijos;

    if (ht == 0)
    {
        for (int j = 0; j < h.m; j++)
        {
            s += indent + hijos[j].llave + " " + hijos[j].valor + "\n";
        }
    }
    else
    {
        for (int j = 0; j < h.m; j++)
        {
            if (j > 0) s += indent + "(" + hijos[j].llave + ")\n";
            s += toString(hijos[j].siguiente, ht-1, indent + "
");
        }
    }
    return s;
}

// Metodo que compara los nodos
private boolean menores(Comparable k1, Comparable k2)

```

```

    {
        return k1.compareTo(k2) < 0;
    }

private boolean iguales(Comparable k1, Comparable k2) {
    return k1.compareTo(k2) == 0;
}

/*****@throws IOException*****/

public static void main(String[] args) throws IOException {
    Arbol_B<String, String> arbol = new Arbol_B<String, String>();

    Scanner stdin = new Scanner(System.in);
    stdin = new Scanner(new File("C:/tst/numeros.txt"));

    File abrir = new File("C:/tst/numeros.txt");
    FileReader leer = new FileReader(abrir);
    stdin = new Scanner(new File("C:/tst/numeros.txt"));

    int y=0;

    //Lee del archivo un entero y una cadena de caracteres
    do
    {
        arbol.agregar(stdin.nextInt(),stdin.next());
    }
    while(stdin.hasNext() != stdin.hasNext("EOF"));
    leer.close();

    System.out.println("\t \t Arbol B \n");
    System.out.println("Busqueda\n" + "los datos que se encuentran en
el
nodo \n");
    System.out.println("3: " + arbol.get(110));
    System.out.println("0: " + arbol.get(0));
    System.out.println("7: " + arbol.get(17));
    System.out.println("5: " + arbol.get(18));
    System.out.println();

    System.out.println("numero de datos: " + arbol.tamaño());
    System.out.println("Altura: " + arbol.Altura());
    System.out.println("\nRecorrido en Orden: \n");
    System.out.println(arbol.toString());
    System.out.println();

    }
}

```