Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Algoritmos y Estructuras de Datos

Tarea 4

Autor: Javier Gómez
Profesor: Benjamín Bustos
Auxiliares: Ignacio Valderrama
Manuel Olguín
Vanessa Peña Araya
Ayudantes: Fabián González
José Ignacio Moreno
Pablo Pizarro R.

Fecha de realización: 19 de noviembre de 2016
Fecha de entrega: 20 de noviembre de 2016
Santiago, Chile

Índice de Contenidos

1. Introducción

En computación hay ocasiones en que es más facil organizar los datos en una estructura denomidada “árboles de búsqueda”, ya sea porque es más eficiente encontrar la información que necesitamos, o porque simplemente la estructura del problema así lo requiere.

Estos árboles “binarios” consisten basicamente en tres partes: nodo, hijo izquierdo e hijo derecho. Todo comienza con una raiz, que estará en el punto más alto del árbol, la cual es un nodo que contiene un valor. Esta tiene dos hijos, los que a su vez también son nodos que tienen un valor y tienen otros dos hijos. Así se va armando un árbol de forma recursiva"(estructuralmente hablando), hasta llegar a un nodo vacío, el cual marca el fin de una rama. La condición es que el valor del hijo izquierdo sea menor que el valor del nodo "padre", y que el valor de hijo derecho sea mayor que el del padre.

Existe una diferencia entre los árboles AVL y los árboles 2-3. Los primeros consisten en árboles que están siempre “balanceados”, es decir, que la diferencia de altura de sus hijos (sub-árboles) no es mayor a una unidad.

Los 2-3 son árboles en dónde todos los nodos internos tienen 2 ó 3 descendientes y todos los nodos hoja tienen la misma longitud o distancia desde la raíz. Los nodos pueden tener uno o dos valores.

2. Diseño de la solución

En este trabajo se creará un programa al cual se le ingresará el tipo de árbol seguido de valores, los cuales se insertarán en el orden de tipeo, para luego devovler el estado final del árbol mediante una representación en String, para luego imprimirla en la salida estándar.

STDIN	STDOUT
AVL 10 20 30 40	20 10 . . 30 . 40 . .
AVL 10 20 30 40 16 14 15	20 14 10 . . 16 15 . . . 30 . 40 . .
2-3 1 2 3 4 5 6	2,4 1 . . 3 . . 5,6 . . .

Ejemplo de input y output

Para esto, implementaremos los árboles del tipo “AVL” y los árboles “2-3”.

3. Implementación

A continuación, se revisará la implementación de los árboles.

3.1. Árboles AVL

Para implementar los árboles, primero se definió la clase *NodoAVL*, que define la estructura de los nodos que componen el árbol.

```
class NodoAVL{
    Comparable valor; //valor del nodo
    NodoAVL izq;
    NodoAVL der;
    int altura;

    //constructor
    public NodoAVL(Comparable valor)
```

```
    {
        this(valor , null, null);
    }

    public NodoAVL(Comparable valor, NodoAVL izq, NodoAVL der){
        this.valor = valor;
        this.izq = izq;
        this.der = der;
        altura = 0; //por defecto
    }
    //creamos un metodo que nos de como string el sub-arbol
    //a partir de un nodo (para despues utilizarla con el nodo 'raiz' del arbol
    public String astr (){
        String aux = "";
        aux+=this.valor+" ";
        if(this.izq == null){
            aux+=". ";}
        else {
            aux+=this.izq.astr();}
        if(this.der == null){
            aux+=". ";}
        else {
            aux+=this.der.astr();}
        return aux;}
}
```

Los nodos cuentan con un valor, un hijo izquierdo (Nodo), un hijo derecho (Nodo), y con un valor "altura" que indica la altura del nodo y su sub-árbol.

Luego, se implementó la clase "ArbolAVL"

```
class ArbolAVL {
    private NodoAVL raiz;

    public void insertar(Comparable x) {
        raiz = insertar(x, raiz);
    }

    private NodoAVL insertar(Comparable x, NodoAVL q) {
        if (q == null)
            q = new NodoAVL(x, null, null);
        else if (x.compareTo(q.valor) < 0) {
            q.izq = insertar(x, q.izq);
            if (alturaH(q.izq) - alturaH(q.der) == 2)
                if (x.compareTo(q.izq.valor) < 0)
                    q = rotarconizq(q);
                else
                    q = doblarizq(q);
        }
        else if (x.compareTo(q.valor) > 0) {
            q.der = insertar(x, q.der);
            if (alturaH(q.der) - alturaH(q.izq) == 2)
                if (x.compareTo(q.der.valor) > 0)
                    q = rotarconder(q);
                else
                    q = doblarder(q);
```

```
        } else
            ;
        q.altura = max(alturaH(q.izq), alturaH(q.der)) + 1;
        return q;
    }

    private static int max(int a, int b) {
        return a > b ? a : b;
    }

    //definimos metodos para poder realizar las rotaciones para el balanceo
    private static NodoAVL rotarconizq(NodoAVL aux2) {
        (...)
    }
    private static NodoAVL rotarconder(NodoAVL aux1) {
        (...)
    }
    private static NodoAVL doblarizq(NodoAVL aux3) {
        (...)
    }
    private static NodoAVL doblarder(NodoAVL aux1) {
        (...)
    }
    //para actualizar altura
    private static int alturaH(NodoAVL q) {
        (...)
    }
}
```

Para ver el detalle de los métodos *rotarconizq, rotarconder, doblarizq, doblarder, alturaH*, ver Anexo 1.

El método *insertar*, además de agregar el nuevo elemento, realiza las comparaciones para determinar si es necesario realizar rotaciones para balancear el árbol, ya que recordemos que la diferencia de altura de los hijos de cada nodo no puede ser mayor a 1 unidad.

Se puede dar cuenta analizando el código, que la inserción es de orden *O(log (n))*, ya que las rotaciones toman tiempo constante y la altura del árbol se limita a *O(log (n))*.

3.2. Árboles 2-3

Para los árboles 2-3, implementamos la clase "ArbolDT" de la siguiente forma:

```
class ArbolDT {
    //puede tener 2 valores, 3 descendientes
    int valor1,valor2;
    ArbolDT izq,mid,der;

    ArbolDT(int val, ArbolDT izq, ArbolDT der)
    {
        this.izq = izq;
        this.der = der;
        this.mid = null;
        this.valor1 = val;
        this.valor2 = val;}

    //constructor (un valor)
    ArbolDT(int val)
```

```
    {this(val,null,null);}

    ArbolDT insertar(int val) {
        (...)
    }

    //convierte el arbol a string
    //(ej. a = arbol.astr; y luego System.out.println(a); imprime el arbol)
    String astr() {
        (...)
    }
```

Se omitió el contenido de los métodos *insertar* y *astr* debido a su extensión. Para mayores detalles ver Anexo 1.

Al igual que en los árboles AVL, en los 2-3 utilizamos comparaciones dentro del método *insertar* para determinar cuándo y cómo se deben realizar las rotaciones para equilibrar el árbol.

Estas operaciones toman en el peor caso tiempo *O(log (n))*.

3.3. Recepción de valores e impresión del árbol

Finalmente, se implementó el método "main" del programa, para recibir los valores por la entrada estándar y luego imprimir el árbol mediante la salida estándar.

```
public class main {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        while(in.hasNextLine()) {
            String sec = in.nextLine();

            String[] secuencia = sec.split(" ");
            int n = secuencia.length;
            String op = secuencia[0];

            if (op.equals("AVL")){
                //crear avl
                ArbolAVL arbol = new ArbolAVL();
                //insertamos los elementos
                for (int i = 1; i<n; ++i){
                    arbol.insertar(Integer.parseInt(secuencia[i]));
                }

                //outeamos el string
                String arbols;
                arbols = arbol.raiz.astr();
                System.out.println(arbols);

            }

            //es 2-3
            else{
                //crear 2-3
                ArbolDT arbol = new ArbolDT(Integer.parseInt(secuencia[1]));
                //insertamos los elementos
                for (int i = 2; i<n; ++i){
```

```
                arbol.insertar(Integer.parseInt(secuencia[i]));
            }
            //outeamos el arbol
            String arbols;
            arbols = arbol.astr();
            System.out.println(arbols);
        }
    }

}
}
```

4. Resultados y Conclusiones

Se puede observar facilmente que la implementación de estos árboles es bastante compleja, pero a pesar de esto, los árboles AVL son considerablemente más faciles de implementar que los árboles 2-3. Sin embargo, aunque los peores casos de cada uno tienen la misma complejidad, los árboles 2-3 son más rápidos a la hora de insertar valores, pero más lentos a la hora de buscar un elemento, por lo que es aquí donde el programador debe cuestionarse la utilidad que le dará al árbol para elegir correctamente el que implementará.

5. Anexo 1: Código del programa

```
import java.util.Scanner;


//PRIMERO IMPLEMENTAMOS ARBOLES AVL

class NodoAVL{
    Comparable valor; //valor del nodo
    NodoAVL izq;
    NodoAVL der;
    int altura;

    //constructor
    public NodoAVL(Comparable valor)
    {
        this(valor , null, null);
    }

    public NodoAVL(Comparable valor, NodoAVL izq, NodoAVL der){
        this.valor = valor;
        this.izq = izq;
        this.der = der;
        altura = 0; //por defecto
    }

    //creamos un metodo que nos de como string el sub-arbol
    //a partir de un nodo (para despues utilizarla con el nodo 'raiz' del arbol
    public String astr (){
        String aux = "";
        aux+=this.valor+" ";
        if(this.izq == null){
            aux+=". ";
        }
        else {
            aux+=this.izq.astr();
        }
        if(this.der == null){
            aux+=". ";
        }
        else {
            aux+=this.der.astr();

        }
        return aux;
    }
}


class ArbolAVL {
    NodoAVL raiz;

    public void insertar(Comparable x) {
        raiz = insertar(x, raiz);
```

```
}


private NodoAVL insertar(Comparable x, NodoAVL q) {
    if (q == null)
        q = new NodoAVL(x, null, null);
    else if (x.compareTo(q.valor) < 0) {
        q.izq = insertar(x, q.izq);
        if (alturaH(q.izq) - alturaH(q.der) == 2)
            if (x.compareTo(q.izq.valor) < 0)
                q = rotarconizq(q);
            else
                q = doblarizq(q);
    }
    else if (x.compareTo(q.valor) > 0) {
        q.der = insertar(x, q.der);
        if (alturaH(q.der) - alturaH(q.izq) == 2)
            if (x.compareTo(q.der.valor) > 0)
                q = rotarconder(q);
            else
                q = doblarder(q);
    } else
        ;
    q.altura = max(alturaH(q.izq), alturaH(q.der)) + 1;
    return q;
}


private static int max(int a, int b) {
    return a > b ? a : b;
}


//definimos metodos para poder realizar las rotaciones para el balanceo


private static NodoAVL rotarconizq(NodoAVL aux2) {
    NodoAVL aux1 = aux2.izq;
    aux2.izq = aux1.der;
    aux1.der = aux2;
    aux2.altura = max(alturaH(aux2.izq), alturaH(aux2.der)) + 1;
    aux1.altura = max(alturaH(aux1.izq), aux2.altura) + 1;
    return aux1;
}


private static NodoAVL rotarconder(NodoAVL aux1) {
    NodoAVL aux2 = aux1.der;
    aux1.der = aux2.izq;
    aux2.izq = aux1;
    aux1.altura = max(alturaH(aux1.izq), alturaH(aux1.der)) + 1;
    aux2.altura = max(alturaH(aux2.der), aux1.altura) + 1;
    return aux2;
}
```

```
    private static NodoAVL doblarizq(NodoAVL aux3) {
        aux3.izq = rotarconder(aux3.izq);
        return rotarconizq(aux3);
    }


    private static NodoAVL doblarder(NodoAVL aux1) {
        aux1.der = rotarconizq(aux1.der);
        return rotarconder(aux1);
    }

    //para actualizar altura
    private static int alturaH(NodoAVL q){
        return q == null ? -1 : q.altura;}

}

//AHORA IMPLEMENTAMOS ARBOLES 2-3

class ArbolDT {
    //puede tener 2 valores, 3 descendientes
    int valor1,valor2;
    ArbolDT izq,mid,der;


    ArbolDT(int val, ArbolDT izq, ArbolDT der)
    {
        this.izq = izq;
        this.der = der;
        this.mid = null;
        this.valor1 = val;
        this.valor2 = val;}

    //constructor (un valor)
    ArbolDT(int val)
    {this(val,null,null);}


    //METODO DE INSERCION EN ARBOL 2-3
    ArbolDT insertar(int val) {
        if(this.izq == null)
        {
            if(this.valor1==this.valor2) //tiene 1 valor
            {
                if(this.valor1<val)
                {
                    this.valor2=val;
                    return null;}
                else
                {
                    this.valor1=val;
                    return null;}
            }
            else //tiene 2 valores
            {
                if(this.valor2<val) //val es mayor a los valores
```

```
        {
            //dos arboles aux (denuevo)
            ArbolDT izq = new ArbolDT(this.valor1);
            ArbolDT der = new ArbolDT(val);
            this.izq = izq;
            this.der = der;
            this.valor1=this.valor2;
            return new ArbolDT(this.valor2,izq,der); //nuevo arbol
            }
        else if(this.valor1>val){ //val es menor a los valores
            //creamos dos arboles auxiliares
            ArbolDT izq=new ArbolDT(val);
            ArbolDT der=new ArbolDT(this.valor2);
            this.izq=izq;
            this.der=der;
            this.valor2=this.valor1;
            return new ArbolDT(this.valor1,izq,der); //creamos nuevo arbol
        }

        else //val es valor intermedio
        {
            //la misma estructura que antes
            ArbolDT izq = new ArbolDT(this.valor1);
            ArbolDT der = new ArbolDT(this.valor2);
            this.izq=izq;
            this.der=der;
            this.valor1=val;
            this.valor2=this.valor1;
            return new ArbolDT(val,izq,der);
        }
    }
}

else
{
    if(this.valor1>val)
    {
        ArbolDT desc = this.izq.insertar(val);
        if(desc!=null) //descenciente no es vacio
        {
            if(this.valor1==this.valor2)
            {
                this.mid=desc.der;
                this.izq=desc.izq;
                this.valor1=desc.valor1;
                return null;}
            else
            {
                //seguimos la misma estructura de crear dos arboles auxiliares para operar
                //y luego crear otro para retornar
                ArbolDT izq = new ArbolDT(desc.valor1,desc.izq,desc.der);
                ArbolDT der = new ArbolDT(this.valor2,this.mid,this.der);
                this.izq = izq;
                this.der = der;
                this.mid = null;
                this.valor2 = this.valor1;
```

```
            return new ArbolDT(this.valor1,izq,der);}
        }
        else //descendiente es vacio
        {
            return null;}
    }
    else if(this.valor2<val)
    {
        ArbolDT desc = this.der.insertar(val);
        if(desc!=null)
        {
            if(this.valor1==this.valor2)
            {
                this.mid=desc.izq;
                this.der=desc.der;
                this.valor2=desc.valor1;

                return null;
            }
            else
            {
                ArbolDT izq = new ArbolDT(this.valor1,this.izq,this.mid);
                ArbolDT der = new ArbolDT(desc.valor1,desc.izq,desc.der);
                this.izq = izq;
                this.der = der;
                this.mid = null;
                this.valor1 = this.valor2;

                return new ArbolDT(this.valor2,izq,der);
            }
        }
        else
        {
            return null;}
    }
    else
    {
        ArbolDT desc = this.mid.insertar(val);
        if(desc!=null)
        {
            ArbolDT izq = new ArbolDT(this.valor1,this.izq,desc.izq);
            ArbolDT der = new ArbolDT(this.valor2,desc.der,this.der);
            this.mid = null;
            this.izq = izq;
            this.der = der;
            this.valor1 = desc.valor1;
            this.valor2 = this.valor1;

            return new ArbolDT(desc.valor1,izq,der);
        }
        else
        {
            return null;
        }

    }
```

```
        }
    }

    //convierte el arbol a string
    //(ej. a = arbol.astr; y luego System.out.println(a); imprime el arbol)
    String astr() {
        //debemos ir recorriendo el arbol
        String aux = ""; //crea un string aux
        if(this.valor1!=this.valor2) //2 valores distintos
        {
            aux+=this.valor1+","+this.valor2+" "; //agrega los 2 valores a aux
        }
        else //1 solo valor
            aux+=this.valor1+" "; //agrega el valor a aux
        if(this.izq!=null)
            aux+=this.izq.astr();
        else //llega al final
            aux+=". ";
        if(this.mid!=null)
            aux+=this.mid.astr();
        else if(this.valor1!=this.valor2)
            aux+=". ";
        if(this.der!=null)
            aux+=this.der.astr();
        else
            aux+=". ";

        return aux;
    }

}

public class main {

    public static void main(String[] args) {
        //CODIGO AQUI

        Scanner in = new Scanner(System.in);
        while(in.hasNextLine()) {
            String sec = in.nextLine();

            String[] secuencia = sec.split(" ");
            int n = secuencia.length;
            String op = secuencia[0];

            if (op.equals("AVL")){
                //crear avl
                ArbolAVL arbol = new ArbolAVL();
                //insertamos los elementos
                for (int i = 1; i<n; ++i){
                    arbol.insertar(Integer.parseInt(secuencia[i]));
                }

                //outeamos el string
                String arbols;
                arbols = arbol.raiz.astr();
```

```
            System.out.println(arbols);

        }

        //es 2-3
        else{
            //crear 2-3
            ArbolDT arbol = new ArbolDT(Integer.parseInt(secuencia[1]));
            //insertamos los elementos
            for (int i = 2; i<n; ++i){
                arbol.insertar(Integer.parseInt(secuencia[i]));
            }
            //outeamos el arbol
            String arbols;
            arbols = arbol.astr();
            System.out.println(arbols);
        }


    }

  }
}
```

Nota: se implementó todo el código en un solo archivo para no tener problema con Moulinette y los packages.