Departamento de Ciencias de la Computación
Facultad de Ciencias Físicas y Matemáticas
Universidad de Chile
CC3001-2 Algoritmos y Estructuras de Datos

ARBOL 2-3

Informe Tarea 4

Integrante: Martin Araya
Profesor: Patricio Poblete
Auxiliares: Daniela Campos
Sven Reisenegger
Ayudantes: Bruno Rodríguez
Cristian Palma
Gabriel Chandía
Tomas Vallejos

Fecha de realización: 30 de noviembre de 2018
Fecha de entrega: 30 de noviembre de 2018
Santiago, Chile

1. Descripción del Problema

El presente informe corresponde al desarrollo de la tarea número cuatro del curso algoritmos y estructuras de datos, tarea consistente en la programación de un diccionario usando la estructura conocida como Árbol 2-3 implementando las operaciones de 'inserción', 'búsqueda', 'altura' e 'imprimir', definidas según el enunciado de la misma tarea.

Un árbol 2-3 corresponde a una estructura de datos similar a un arbol binario de búsqueda, con la diferencia de que en este caso cada nodo puede contener 2 valores (o llaves) y 3 hijos, cumpliéndose siempre la condición de que todos los valores del hijo izquierdo son menores que el valor izquierdo, todos los valores contenidos en el hijo central están entre el valor izquierdo y el valor derecho, y todos los valores contenidos en el hijo derecho son mayores que el valor izquierdo; por último, el valor izquierdo debe ser menor al valor derecho, una visualización de un arbol 2-3 es la figura:

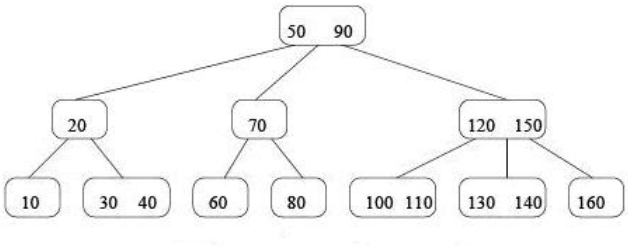

Figura 1: Ejemplo Árbol 2-3

La interacción con el árbol debe ser realizada mediante la entrada estándar, donde se debe introducir una línea con las operaciones a realizar, separadas por un espacio, las operaciones están descritas de la siguiente forma:

- 'inserción': el comando esá determinado de la forma $+ke$, siendo k la llave del elemento e, se debe imprimir 'Error' en la salida estándar si el elemento fue agregado previamente.
- 'búsqueda': el comando está definido de la forma $?k$ donde k es la llave del elemento e a buscar, si el elemento existe en el árbol, este debe ser impreso, en caso contrario, se retorna 'Error'
- 'altura': el comando está definido por h y se debe retornar la altura del árbol al momento de recibir el comando
- 'imprimir': el comando está definido por p y se debe imprimir el árbol en su totalidad en la notación $infix$

Por último, el caso de borde encontrado en este caso es la impresion de un árbol que no presenta elementos en su interior.

2. Descripción de la solución

Para solucionar la Tarea, se implementaron dos clases en java, una correspondiente a los nodos del arbol, encargada de las operaciones pertinentes a estos, y otra encargada de la implementación del arbol, junto con la interfaz del ususario asociada, las funciones incorporadas en cada caso son las siguientes, obviando generadores, getters y setters.

Luego, para realizar cada una de las acciones requeridas en la tarea se utilizaron las siguientes combinaciones de operaciones:

2.1. 'Inserción'

Para realizar la inserción de un elemento xx con llave x se realiza lo siguiente:

- Primero, se calcula si el elemento ya existe previamente en el árbol, ocupando la función findItem(Tree23Node, int)
  - Si el elemento no existe en el arbol, se continua
    - Si la raiz del arbol es nula, se crea el arbol con el elemento y la llave requeridas
    - Si no, se calcula el nodo donde el elemento estaría y calculando la lista de padres, mediante el metodo getLeafNode(Tree23Node,int), y se inserta el elemento en el nodo, usando insertItem(int,String)
    - Finalmente, si el nodo queda con 3 elementos (overflow) se divide el nodo usando splitNode(Tree23Node)
  - Si el elemento ya existe en el arbol, se imprime 'Error'

2.2. 'Búsqueda'

Para realizar la búsqueda, e impresion, del elemento con llave x se realiza lo siguiente

- Primero, se encuentra el nodo donde está el elemento de llave x dentro del árbol mediane el metodo findItem(Tree23Node,int)
  - Si la llave no está en el árbol, se imprime 'Error' y se acaba el método
  - Si la llave está dentro del árbol, se compara con cada uno de los valores del nodo, y se imprime el elemento correspondiente

2.3. 'Altura'

Para mostrar la altura del árbol, tomamos en cuenta la propiedad de los arboles 2-3 que hace que todos los hijos se encuentren al mismo nivel de profundidad, por ende, reducimos el cálculo de la altura solo a el lado izquierdo del árbol, por ende, solo hacemos el llamado al hijo izquierdo

- Si la raiz del árbol es nula, la altura es 0
- Se crea un nodo auxiliar para evitar problemas con los punteros
- se recorre hacia la izquierda aumentando un contador hasta encontrar un nodo que sea hoja, imprimiendo la altura y terminando el método

2.4. 'Imprimir'

Para este método, se recorre el árbol, imprimiendo cada uno de los valores del nodo al ir 'pasando' por este

- Si el nodo es hoja
  - Se imprimen los valores del nodo, con el método printLeaf()
  - Se termina el método
- Se imprime el hijo derecho ocupando el método recursivamente
- Se imprime la llave menor
- Se imprime el hijo central ocupando el método recursivamente
- Si existe una tercera llave
  - Se imprime la llave mayor
  - Se imprime el hijo mayor ocuando el método recursivamente

De esta forma, se tienen todas las operaciones necesarias para realizar todas las acciones requeridas.

3. Resultados

Para los resultados, se proó cada una de las operaciones por separado, imprimiendo el árbol en cada una, para demostrar el correcto funcionamiento, luego se probaron combinaciones de operaciones.

3.1. Operaciones por separado

3.1.1. Insertar

El comando seleccionado trata de insertar dos veces el mismo elemento, con la misma llave, y una vez un elemento distinto, pero igualmente con la misma llave, generándose dos veces un error:

Código 1: Ejemplo insertar

```
+1a +1a +1b p
Error Error ([]1[])
```

3.1.2. Búsqueda

El comando seleccionado crea un árbol simple de 3 elementos, lo imprime, luego genera una busqueda correcta de uno de ellos, para finalizar en una búsqueda infructuosa

Código 2: Ejemplo búsqueda

```
+1a +2b +3c p ?2 ?4
 (([]1[]) 2([]3[]) ) b Error
```

3.1.3. Altura

El comando seleccionado va agregando elementos al árbol e imprimiendo la áltura del mismo en cada paso, imprimiendo el árbol al final para comprobar la altura

Código 3: Ejemplo altura

```
h +1a h +2b h +3c h +4d h +5e h +6f h +7g h p
0  1  1  2  2  2  2  3   ((([]1[])  2([]3[]) ) 4(([]5[])  6([]7[]) ))
```

3.2. Imprimir

En los ejemplos anteriores se ha visto la impresión de forma correcta para cada uno de los casos, ahora, si tomamos el caso de borde de imprimir un árbol vacío, tenemos:

Código 4: Ejemplo Imprimir

```
p
 ([][])
```

Anexo A. Código Fuente

Código A.1: Código Nodos

```
public class Tree23Node {
    //La clase contiene un parametro para cada uno de los valores que se pueden guardar en el nodo, ademas de valores
    //auxiliares que nos ayudan en caso de overflow
    private int smallitem;
    private String smallElement;
    private int largeitem;
    private String largeElement;
    private Tree23Node leftChild;
    private Tree23Node midChild;
    private Tree23Node rightChild;

    private int auxItem;
    private String auxElement;
    private Tree23Node auxChild;

    // Constructor de un nodo con un valor aa y un elemento aa
    public Tree23Node(int a, String aa){
        smallitem = a; smallElement = aa;
        largeitem = auxItem = Integer.MAX_VALUE;
        leftChild = midChild = rightChild = null;
    }

    //Constructor de un nodo con dos elementos, elemento aa con valor a y elemento bb con valor b
    public Tree23Node(int a, int b, String aa, String bb){
        smallitem = a; smallElement = aa;
        largeitem = b; largeElement = bb;
        auxItem = Integer.MAX_VALUE;
        leftChild = midChild = rightChild = null;
    }

    //Constructor de un nodo sin elementos
    public Tree23Node(){
        smallitem = largeitem = auxItem = Integer.MAX_VALUE;
        leftChild = rightChild = midChild = null;
    }

    //isLeaf(): metodo que determina si el nodo consultado corresponde a una hoja
    public boolean isLeaf(){
        return (leftChild == null && midChild == null && rightChild == null);
    }

    //nodeCount(): metodo para calcular el numero de nodos hijos pertenecientes al nodo, segun los valores almacenados
    //en el
    public int nodeCount(){
        if (smallitem == Integer.MAX_VALUE){
            return 0;
        }
        if (largeitem == Integer.MAX_VALUE){
            return 1;
        }
        if(auxItem == Integer.MAX_VALUE){
            return 2;
        }
        return 3;
    }

    //metodo para imprimir los valores del nodo sobre el cual se llama el metodo, asumiendo que este es una hoja
    public void printLeaf(){
        System.out.print("[]" + smallitem + "[]");
        if(largeitem != Integer.MAX_VALUE){
            System.out.print(largeitem + "[]");
        }
    }

    //Getters and setters de la clase
    public String getSmallElement() {
        return smallElement;
    }

    public String getLargeElement() {
        return largeElement;
```

```
72      }
73
74      public String getAuxElement() {
75          return auxElement;
76      }
77
78      public int getSmallitem() {
79          return smallitem;
80      }
81
82      public int getLargeitem() {
83          return largeitem;
84      }
85
86      public Tree23Node getLeftChild() {
87          return leftChild;
88      }
89
90      public Tree23Node getMidChild() {
91          return midChild;
92      }
93
94      public Tree23Node getRightChild() {
95          return rightChild;
96      }
97
98      public int getAuxItem() {
99          return auxItem;
100     }
101
102     public Tree23Node getAuxChild() {
103         return auxChild;
104     }
105
106     public void setLeftChild(Tree23Node leftChild) {
107         this.leftChild = leftChild;
108     }
109
110     public void setMidChild(Tree23Node midChild) {
111         this.midChild = midChild;
112     }
113
114     public void setRightChild(Tree23Node rightChild) {
115         this.rightChild = rightChild;
116     }
117
118     public void setAuxChild(Tree23Node auxChild) {
119         this.auxChild = auxChild;
120     }
121
122     //insertItem(int String): se inserta el elemento xx con valor (o llave) x en el nodo sobre el cual se llama la
123     //funcion, teniendo un comportamiento distinto segun el numero de valores que el nodo ya contiene, manteniendo
124     //la relacion de valor de los valores dentro del nodo
125     public void insertItem(int x, String xx){
126         if(smallitem == Integer.MAX_VALUE){
127             smallitem = x;
128             smallElement = xx;
129         }
130         else if(largeitem == Integer.MAX_VALUE){
131             if(x>smallitem){
132                 largeitem = x;
133                 largeElement = xx;
134             }
135             else {
136                 largeitem = smallitem;
137                 largeElement = smallElement;
138                 smallitem = x;
139                 smallElement = xx;
140             }
141         }
142         else{
143             int [] lista = new int[3];
144             String [] lista2 = new String[3];
145             if (smallitem<largeitem){
146                 if(x<smallitem){
147                     lista [0] = x;
148                     lista2 [0] = xx;
149                     lista [1] = smallitem;
```

```
150                     lista2 [1] = smallElement;
151                     lista [2] = largeitem;
152                     lista2 [2] = largeElement;
153                 }
154                 else{
155                     if(x<largeitem) {
156                         lista [0] = smallitem;
157                         lista2 [0] = smallElement;
158                         lista [1] = x;
159                         lista2 [1] = xx;
160                         lista [2] = largeitem;
161                         lista2 [2] = largeElement;
162                     }
163                     else{
164                         lista [0] = smallitem;
165                         lista2 [0] = smallElement;
166                         lista [1] = largeitem;
167                         lista2 [1] = largeElement;
168                         lista [2] = x;
169                         lista2 [2] = xx;
170                     }
171                 }
172             }
173             else{
174                 if(x<largeitem){
175                     lista [0] = x;
176                     lista2 [0] = xx;
177                     lista [1] = largeitem;
178                     lista2 [1] = largeElement;
179                     lista [2] = smallitem;
180                     lista2 [2] = smallElement;
181                 }
182                 else{
183                     if (x<smallitem) {
184                         lista [0] = largeitem;
185                         lista2 [0] = largeElement;
186                         lista [1] = x;
187                         lista2 [1] = xx;
188                         lista [2] = smallitem;
189                         lista2 [2] = smallElement;
190                     }
191                     else{
192                         lista [0] = largeitem;
193                         lista2 [0] = largeElement;
194                         lista [1] = smallitem;
195                         lista2 [0] = smallElement;
196                         lista [2] = x;
197                         lista2 [2] = xx;
198                     }
199                 }
200             }
201             smallitem = lista [0];
202             smallElement = lista2 [0];
203             largeitem = lista [1];
204             largeElement = lista2 [1];
205             auxItem = lista [2];
206             auxElement = lista2[2];
207         }
208     }
209
210     //splitNode(Tree23Node): se divide el nodo sobre el cual se ejecuta el metodo, el comportamiento varia segun la
211     //relacion al nodo padre, que se incorpora en los argumentos del metodo
212     public void splitNode(Tree23Node p){
213         Tree23Node l = new Tree23Node(smallitem, smallElement);
214         Tree23Node r = new Tree23Node(auxItem, auxElement);
215
216         if(!this.isLeaf()){
217             l.setLeftChild(this.getLeftChild());
218             l.setMidChild(this.getMidChild());
219             r.setLeftChild(this.getRightChild());
220             r.setMidChild(this.getAuxChild());
221         }
222         if(this == p.getLeftChild()){
223             if(p.nodeCount() == 1){
224                 p.setRightChild(p.getMidChild());
225                 p.setMidChild(r);
226                 p.setLeftChild(l);
227             }
```

```
            else if(p.nodeCount() == 2){
                p.setAuxChild(p.getRightChild());
                p.setRightChild(p.getMidChild());
                p.setMidChild(r);
                p.setLeftChild(l);
            }
        }
        else if(this == p.getMidChild()){
            if(p.nodeCount() == 1){
                p.setRightChild(r);
                p.setMidChild(l);
            }
            else if(p.nodeCount() == 2){
                p.setAuxChild(p.getRightChild());
                p.setRightChild(r);
                p.setMidChild(l);
            }
        }
        else if(this == p.getRightChild()){
            p.setAuxChild(r);
            p.setRightChild(l);
        }
        else{
            p.setLeftChild(l);
            p.setMidChild(r);
        }
    }
}
```

Código A.2: Código Arbol e interfaz

```
import java.util.ArrayList;
import java.util.Scanner;

public class Tree23 {
    //la clase contiene 2 parametros, la raiz del arbol Root, que corresponde a un nodo y el ArrayLista parentList que
    //permite recopilar los padres de un nodo especifico
    private Tree23Node Root;
    private ArrayList<Tree23Node> parentList = new ArrayList<>();

    //constructor crea un arbol con raiz null, al cual se le aplicaran las funciones
    private Tree23(){
        Root = null;
    }

    //heihgt(): calcula la altura del arbol consultado, a partir de su raiz, ya que una propiedad de los arboles 2-3 es
    //que todos sus hijos tienen la misma altura, se recorren solo los hijos del lado izquierdo
    private void height(){
        if (this.Root == null){
            System.out.print(0 + " ");
            return;
        }
        Tree23Node aux = this.Root;
        int i = 1;
        while(true) {
            if (aux.isLeaf()) {
                System.out.print(i + " ");
                return;
            }
            i++;
            aux = aux.getLeftChild();
        }
    }

    //inorder(Tree23Node): imprime el arbol en la notacion infix, con los parentesis correspondientes, si el nodo con-
    //sultado es una hoja, se llama al metodo para imprimir los valores de la hoja, en caso contrario, se recorren los
    //hijos y valores de forma ascendente, en el caso de que el nodo sea nulo, se imprime un arbol vacio
    private void inorder(Tree23Node root){
        if (root == null){
            System.out.print(" ([][]) ");
            return;
        }
        System.out.print("(");
        if(root.isLeaf()){
            root.printLeaf();
            System.out.print(")");
```

```
46              return;
47          }
48          inorder(root.getLeftChild());
49          System.out.print(root.getSmallitem());
50          inorder(root.getMidChild());
51          if(root.nodeCount() == 2){
52              System.out.print(root.getLargeitem());
53              inorder(root.getRightChild());
54          }
55          System.out.print(") ");
56      }
57
58      //getLeafNode(Tree23Node int): retorna el nodo que corresponderia al elemento de valor (o llave) x, agregando los
59      //nodos recorridos a la lista de padres
60      private Tree23Node getLeafNode(Tree23Node root, int x){
61          if(root == null || root.isLeaf()){
62              return root;
63          }
64          parentList.add(root);
65          if(x<root.getSmallitem()){
66              return getLeafNode(root.getLeftChild(),x);
67          }
68          if(root.getLargeitem() == Integer.MAX_VALUE || x<root.getLargeitem()) {
69              return getLeafNode(root.getMidChild(), x);
70          }
71          return getLeafNode(root.getRightChild(),x);
72      }
73
74      //findItem(tree23Node, int): retorna el nodo donde esta el elemento de valor(o llave) x, recorriendo el arbol desde
75      //los valores menores a los mayores, pasando por los hijos
76      private Tree23Node findItem(Tree23Node root, int x){
77          if(root == null){
78              return null;
79          }
80          if (root.isLeaf()){
81              if(x==root.getSmallitem() || (root.nodeCount() == 2 && x == root.getLargeitem())){
82                  return root;
83              }
84              return null;
85          }
86          if(x<root.getSmallitem()){
87              return findItem(root.getLeftChild(),x);
88          }
89          if(x == root.getSmallitem()){
90              return root;
91          }
92          if(root.nodeCount() == 1 || x<root.getLargeitem()){
93              return findItem(root.getMidChild(),x);
94          }
95          if(x == root.getLargeitem()){
96              return root;
97          }
98          return findItem(root.getRightChild(),x);
99      }
100
101     //printItem(tree23Node, int): Imprime el elemento correspondiente al valor (o llave) x, siempre que el elemento ya
102     //este en el arbol, en caso contrario, imprime "Error"
103     private void printItem(Tree23Node root, int x){
104         Tree23Node nodo = findItem(root,x);
105         if(nodo == null){
106             System.out.print("Error ");return;
107         }
108         if(x == nodo.getSmallitem()){
109             System.out.print(nodo.getSmallElement()+ " ");return;
110         }
111         if(nodo.nodeCount() == 2 && x==root.getLargeitem()){
112             System.out.print(nodo.getLargeElement()+ " ");
113         }
114     }
115
116     //insert(int, String): inserta el elemento xx con el valor (o llave) x, si la llave ya existe en el arbol, se
117     //imprime "Error"
118     private void insert(int x,String xx){
119         if(findItem(Root,x) != null){
120             System.out.print("Error ");
121             return;
122         }
123         if(Root == null){
```

```
124             Root = new Tree23Node(x,xx);
125             return;
126         }
127         parentList.clear();
128         Tree23Node leaf = getLeafNode(Root,x);
129         leaf.insertItem(x,xx);
130         if(leaf.nodeCount() == 3){
131             splitNode(leaf);
132         }
133     }
134 
135     //splitNode(Tree23Node): se divide el nodo root, calculando el nodo padre de este
136     private void splitNode(Tree23Node root){
137         Tree23Node p;
138         if(root == this.Root){
139             this.Root = p = new Tree23Node();
140         }
141         else{
142             p = parentList.get(parentList.size()-1);
143             parentList.remove(parentList.size()-1);
144         }
145         root.splitNode(p);
146         p.insertItem(root.getLargeitem(),root.getLargeElement());
147         if(p.nodeCount() == 3){
148             splitNode(p);
149         }
150     }
151 
152     public static void main(String[] args){
153         Scanner sc = new Scanner(System.in);
154         Tree23 tree = new Tree23();
155         while(sc.hasNext()){
156             String a = sc.nextLine();
157             String [] b = a.split(" ");
158             for(int i =0; i<b.length;i++){
159                 if (b[i].charAt(0) == 'h'){
160                     tree.height();
161                     System.out.print(" ");
162                 }
163                 else if (b[i].charAt(0) == 'p'){
164                     tree.inorder(tree.Root);
165                 }
166                 else if (b[i].charAt(0) == '+'){
167                     int x = Integer.parseInt(b[i].substring(1,b[i].length()-1));
168                     String xx = b[i].substring(b[i].length()-1);
169                     tree.insert(x,xx);
170                 }
171                 else if (b[i].charAt(0) == '?'){
172                     int x = Integer.parseInt(b[i].substring(1));
173                     tree.printItem(tree.Root,x);
174                 }
175             }
176             System.out.println();
177         }
178     }
179 }
```