Universidad Carlos III de Madrid

Algorithms and Data Structures (ADS)

Bachelor in Informatics Engineering
Computer Science Department

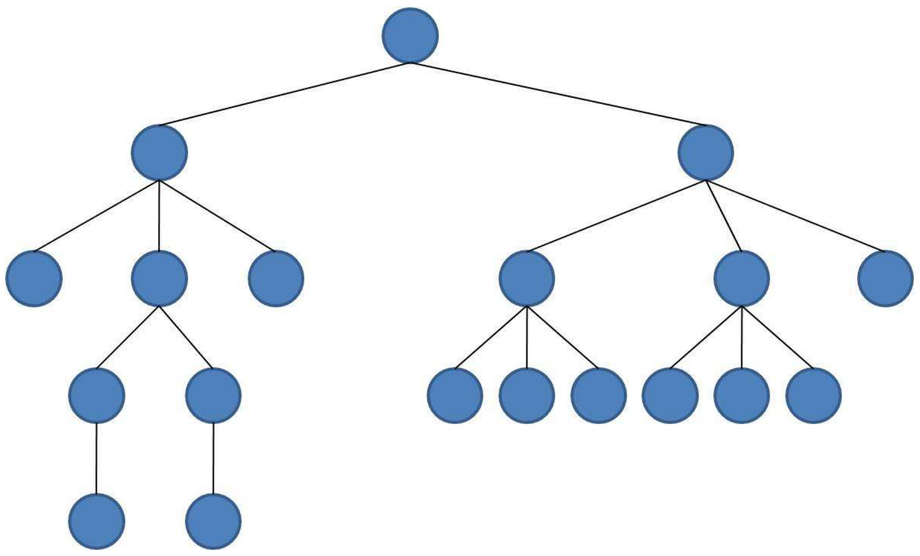
YEAR: 1º / SEMESTER: 2º

EXERCISES SHEET n2 (SOLUTION)

Authors:
Julian Moreno Schneider
Harith Aljumaily
Isabel Segura-Bedmar
Juan Perea

Junio 2011

E-1		



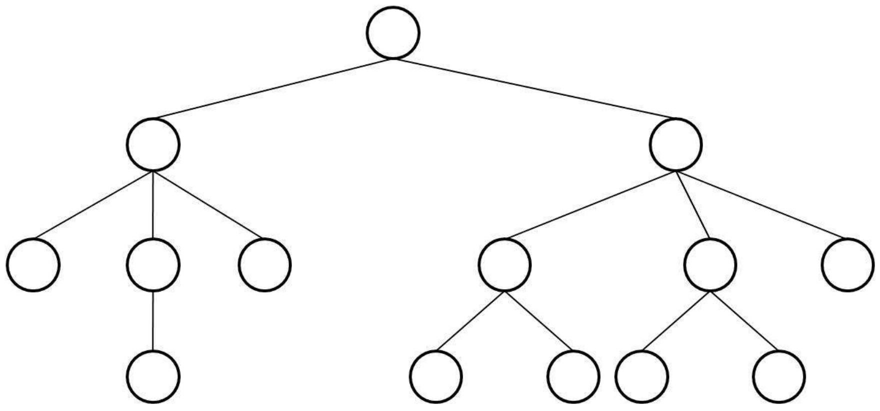
a) Explain the values of the main characteristics of the tree shown in the figure. NOTE: These characteristics are grade of the tree, height, number of nodes, sheets and internal nodes.

b) Given the following properties of a tree, draw a tree that satisfies them:
   1. Grade of the tree: 3
   2. Number of nodes: 14
   3. Height of the tree: 3
   4. Number of nodes with depth=2: 6

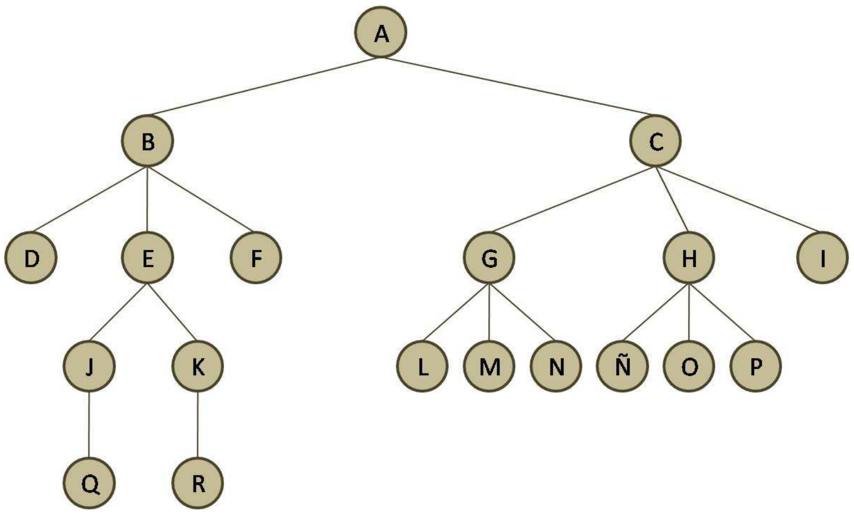
ANSWER:

a) Grade of the tree: 3
   Altura del árbol: 4
   Nodos del árbol: 19
   Nodos Internos: 8
   Nodos externos: 11

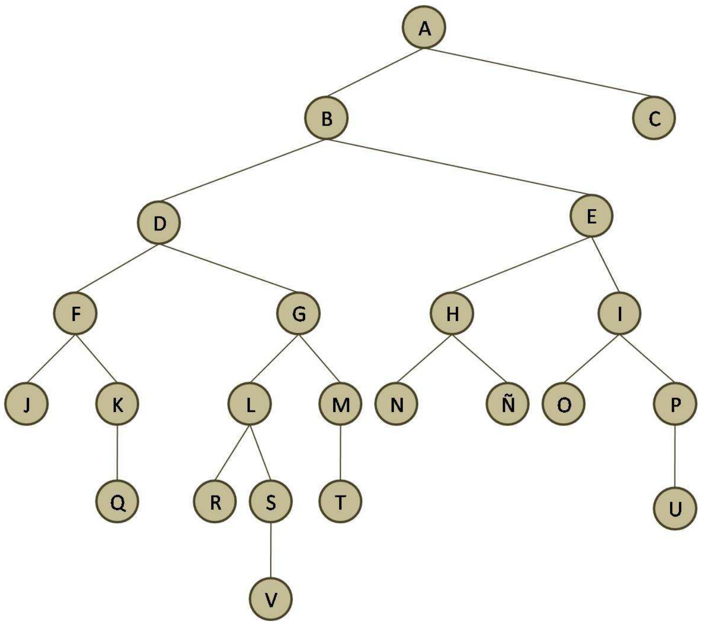
b)



E-2		
Given the following trees, write the pre-order, in-order and post-order traversals of each one.		



a)



b)

ANSWER:

a) Pre-order: ABDEJQKRFCGLMNHÑOPI
In-order: It has not because it is not binary.
Post-order: DQJRKEFBLMNGÑOPHICA

b) Pre-order: ABDFJKQGLRSVMTEHNÑIOPUC
In-order: JFQKDRLVSGTMBNHÑEOIUPAC
Post-order: JQKFRVSLTMGDNÑHOUPIEBCA

E-3		
Implement a java method that returns the máximum value stored in a binary tree of integers. It should return -1 if the tree is empty. **SOLUCIÓN:**		

```java
/**
 * E-3
 */
public int maximunValue(){
	if(this.isEmpty()){
		System.out.println("Tree empty");
		return -1;
	}
	int max = 0;
	BinaryNode<E> aux = this.root;
	while(aux!=null){
		if(aux.value instanceof Integer){
			if((Integer)aux.value>max){
				max = ((Integer)aux.value).intValue();
			}
		}
		else{
			System.out.println("Only applicable to Integer 
values Tree");
			return -1;
		}
		aux = this.preorderNext(aux);
	}
	return max;
}
```

E-4		

Describe in java code an algorithm to compute the number of descendents of each node in a binary tree. The algorithm has to be based on recursivity.

ANSWER:

```java
/**
 * Exercise E-4
 */
public int calculateDescendents(BinaryNode<E> n){
	int right = 0;
	int left = 0;
	if(n.leftChild!=null){
		left = calculateDescendents(n.leftChild)+1;
	}
	if(n.rightChild!=null){
		right = calculateDescendents(n.rightChild)+1;
	}
	return (left+right);
}
```

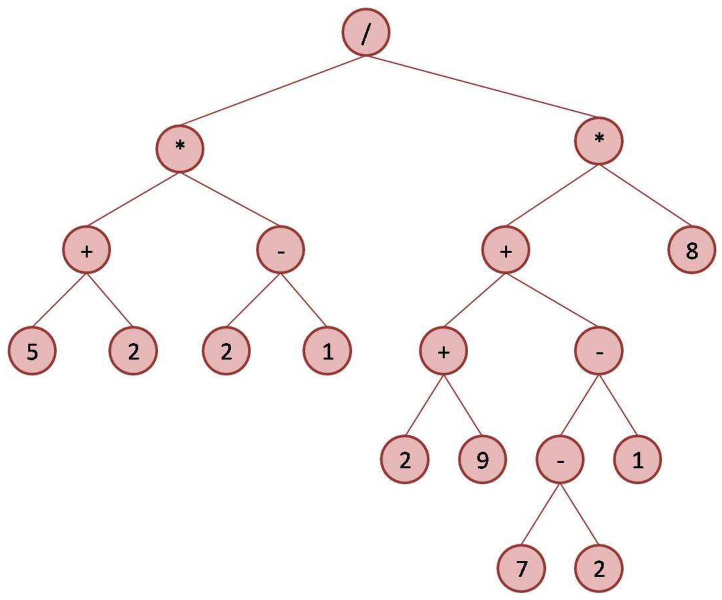
E-5		

Draw the binary tree that represents the following arithmetic expression:

“(((5 + 2) * (2 – 1))/((2 + 9) + ((7 – 2) – 1) * 8)”

Implement in java code the method that makes the representation of the arithmetic expression ('parenthesis included') using the tree implemented in the exercise E-3.

ANSWER:



```java
/**
 * Exercise E-5
 */
public String getExpression(BinaryNode<E> n){
      String s = "";
      if(n.leftChild==null || n.rightChild==null){
            s = n.value.toString();
            return s;
      }
      else{
            s = s.concat("(");
            s = s.concat(getExpression(n.leftChild));
            s = s.concat(n.value.toString());
            s = s.concat(getExpression(n.rightChild));
            s = s.concat(")");
            return s;
      }
}
```

E-6		

Design algorithms in java for the following operations that will be applied to a binary tree:

a) preorderNext(v): returns the next node visited after v using a pre-order traversal.
b) inorderNext(v): returns the next node visited after v using an in-order traversal.
c) postorderNext(v): returns the next node visited after v using an post-order traversal.

Which is the worst case considering execution times of each algorithm?

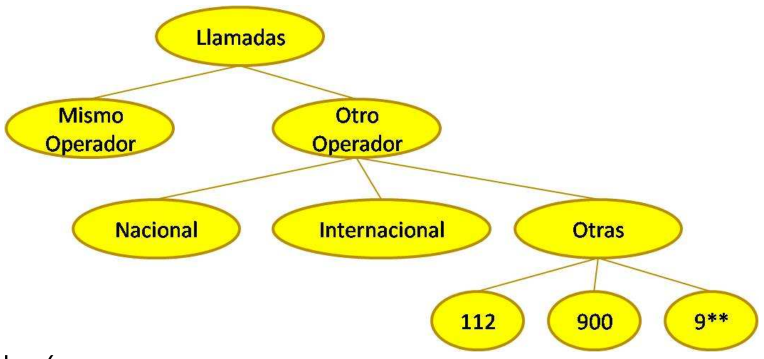
ANSWER:

```java
/**
 * Exercise E-6 a
 */
public BinaryNode<E> preorderNext(BinaryNode<E> v){
      if(v.leftChild!=null){
            return v.leftChild;
	}
	else if(v.rightChild!=null){
		return v.rightChild;
	}
	else{
		BinaryNode<E> auxPadre = v.parent;
		BinaryNode<E> auxHijo = v;
		while(auxPadre!=null && (auxPadre.rightChild==null ||
auxPadre.rightChild.equals(auxHijo))){
			auxHijo = auxPadre;
			auxPadre = auxHijo.parent;
		}
		if(auxPadre==null){
			return null;
		}
		return auxPadre.rightChild;
	}
}

/**
 * Exercise E-6 b
 */
public BinaryNode<E> inorderNext(BinaryNode<E> v){
	if(v.rightChild!=null){
		BinaryNode<E> aux = v.rightChild;
		while(aux.leftChild!=null){
			aux = aux.leftChild;
		}
		return aux;
	}
	else{
	   if(v.parent==null){
		  return null;
	   }
	   if(v.parent.leftChild!=null && v.parent.leftChild.equals(v)){
			return v.parent;
	   }
	   else{
			BinaryNode<E> auxPadre = v.parent;
			BinaryNode<E> auxHijo = v;
			while(auxPadre!=null && (auxPadre.rightChild==null
|| auxPadre.rightChild.equals(auxHijo))){
				auxHijo = auxPadre;
				auxPadre = auxHijo.parent;
			}
			if (auxPadre==null) return null;
			return auxPadre;
		}
	}
}

/**
 * Exercise E-6 c
 */
public BinaryNode<E> postorderNext(BinaryNode<E> v){
	if(v.parent==null){
		return null;
	}
	if(v.parent.rightChild!=null && v.parent.rightChild.equals(v)){
		return v.parent;
	}
```

```
      else{
            if(v.parent.rightChild==null){
                  return v.parent;
            }
            else{
                  BinaryNode<E> aux = v.parent.rightChild;
                  while(aux.isInternal()){
                        if(aux.leftChild!=null){
                              aux = aux.leftChild;
                        }
                        else{
                              aux = aux.rightChild;
                        }
                  }
                  return aux;
            }
      }
}
```

E-7		

In the following figure is shown an example of a tree A and the indented with parenthesis representation of it.



```
      Llamadas(
            Mismo Operador
            Otro Operador(
                  Nacional
                  Internacional
                  Otras(
                        112
                        900
                        9**
                  )
            )
      )
```

a) Implement the algorithm in java code using the tree class implemented in exercise E-3.

ANSWER:

```
/**
 * Exercise E-7
 */
public void represent(GenericNode<E> n,String indent){
      System.out.print(indent + n.value);
```

```java
      if(n.hasChilds()){
            System.out.print("\n");
            System.out.print("\n");
            for(int i=0;i<n.childs.size();i++){
                  represent(n.childs.get(i),indent + "\t");
            }
            System.out.print(indent + ")");
            System.out.print("\n");
      }
      else{
            System.out.print("\n");
      }
}
```

E-8		

Balance Factor of an internal node v of a binary tree is the difference between the height of its left subtree and right subtree. Describe a method to print the balance factors of every internal node of a binary tree using an in-order traversal.

a) Java Code.

ANSWER:

```java
/**
 * Exercise E-8
 */
public void getFactors(BinaryNode<E> n){
      if(n==null){
            return;
      }
      if(n.leftChild!=null){
            getFactors(n.leftChild);
      }
      System.out.println(getHeight(n.leftChild)-
getHeight(n.rightChild));
      if(n.rightChild!=null){
            getFactors(n.rightChild);
      }
}
/**
 * Auxiliar Method used in Exercise E-8
 */
public int getHeight(BinaryNode<E> n){
      if(n==null){
            return 0;
      }
      if(n.leftChild==null && n.rightChild==null){
            return 1;
      }
      int height = 0;
      int leftHeight = getHeight(n.leftChild);
      int rightHeight = getHeight(n.rightChild);
      height = (leftHeight>rightHeight ? leftHeight : rightHeight);
      height ++;
      return height;
}
```

E-9		

Describe an algorithm that gets the root of a binary tree as input and prints the value of the nodes using a level traversal. First level prints the root, next prints all nodes of level 1, then all nodes of level 2, and so on.
NOTA: Pre-order traversals use stacks to do recursive calls. Consider using auxiliary data structures as help to print the level-traversal.

ANSWER:

```
/**
 * Exercise E-9
 */
public void printTreeByLevel(BinaryNode<E> root){
      List<BinaryNode<E>> nodesToWalk=new LinkedList<BinaryNode<E>>();
      nodesToWalk.add(root);
      List<BinaryNode<E>> nodesAux = new LinkedList<BinaryNode<E>>();
      while(!nodesToWalk.isEmpty()){
            nodesAux = new LinkedList<BinaryNode<E>>();
            for(int i=0;i<nodesToWalk.size();i++){
                  BinaryNode<E> nb = nodesToWalk.get(i);
                  System.out.print(nb.value.toString() + " ");
                  if(nb.hasLeft()){
                        nodesAux.add(nb.leftChild);
                  }
                  if(nb.hasRight()){
                        nodesAux.add(nb.rightChild);
                  }
            }
            nodesToWalk = nodesAux;
            System.out.println("");
      }
}
```

E-10		

Having a binary search tree (BST) that stores integers in its nodes. Describe (in java code) a recursive method that adds all the values of the tree nodes.

ANSWER:

```
/**
 * Ejercicio E-10
 */
public int getAddition(BinaryNode<Integer> v){
      if(v.leftChild==null && v.rightChild==null){
            return v.value;
      }
      else if(v.leftChild==null){
            return v.value+getAddition(v.rightChild);
      }
      else if(v.rightChild==null){
            return v.value+getAddition(v.leftChild);
      }
      else{
            return v.value + getAddition(v.leftChild) +
getAddition(v.rightChild);
      }
}
```

E-11		

Having a binary search tree (BST) that stores integers in its nodes. Describe (in java code) a recursive method that prints all values of each node whose grandfather has a value multiple of 5.

ANSWER:

```
/**
 * Ejercicio E-11
 */
public void print5Grandparent(BinaryNode<Integer> v){
      if(v.parent!=null && v.parent.parent!=null &&
(v.parent.parent.value%5==0)){
            System.out.println(v.value);
      }
      if(v.leftChild!=null){
            print5Grandparent(v.leftChild);
      }
      if(v.rightChild!=null){
            print5Grandparent(v.rightChild);
      }
}
```

E-12		

Implement a java method that computes the arythmetic mean of the depths of all the external nodes of a binary tree.

ANSWER:

```
public float getAverageDepth(){
      int depths = 0;
      int nodes = 0;
      BinaryNode<E> aux = root;
      while(aux!=null){
            if(aux.isExternal()){
                  depths +=getDepth(aux);
                  nodes += nodes;
            }
            aux = preorderNext(aux);
      }
      return ((float)depths/(float)nodes);
}
```

E-13		

Having a binary tree that stores String representing the names of the employees of a company. The different levels of the tree represent the different categories of the employees inside the company.
Implement a java method that gets at the input a name and a category and returns the number of employees that have the same name and are hired at this category.

ANSWER:

```
public int getNumberNamedEmployees(String name, int category){
      int number = 0;
      BinaryNode<String> aux = root;
      while(aux!=null){
            if(getDepth(aux)==category && aux.value.equals(name)){
```

```
            number++;
        }
        aux = preorderNext(aux);
    }
    return number;
}
```

E-14		

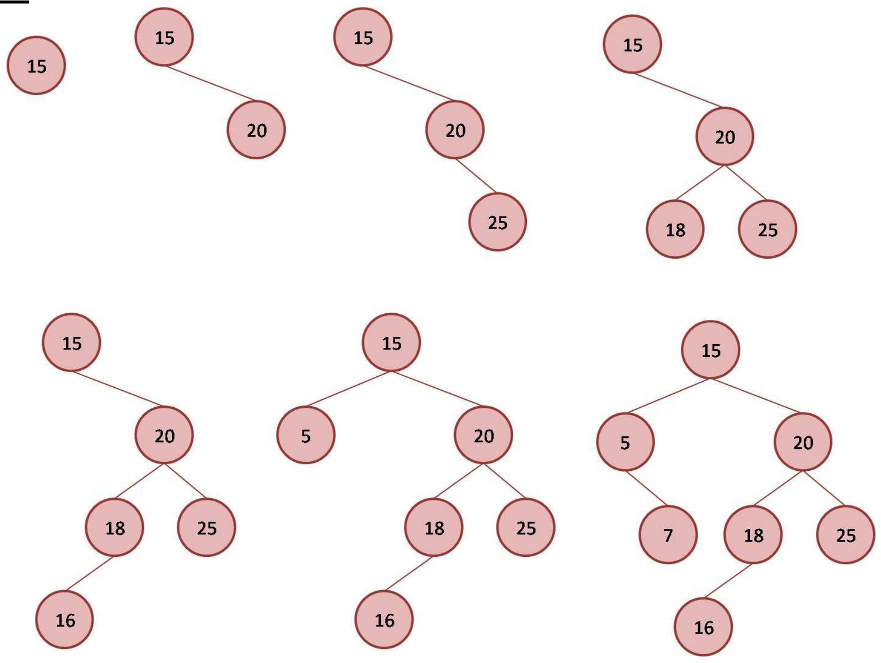
a) Draw the resultant BST of inserting the values 15, 20, 25, 18, 16, 5 y 7 in this order. Draw also the intermediate trees resulting after each insertion.

b) Write the pre-order, in-order and post-order traversal of the resulting tree from part a).

c) Draw the BST completely balanced resulting from inserting 15, 20, 25, 18, 16, 5 y 7 in this order. Draw also the intermediate trees resulting after each insertion.

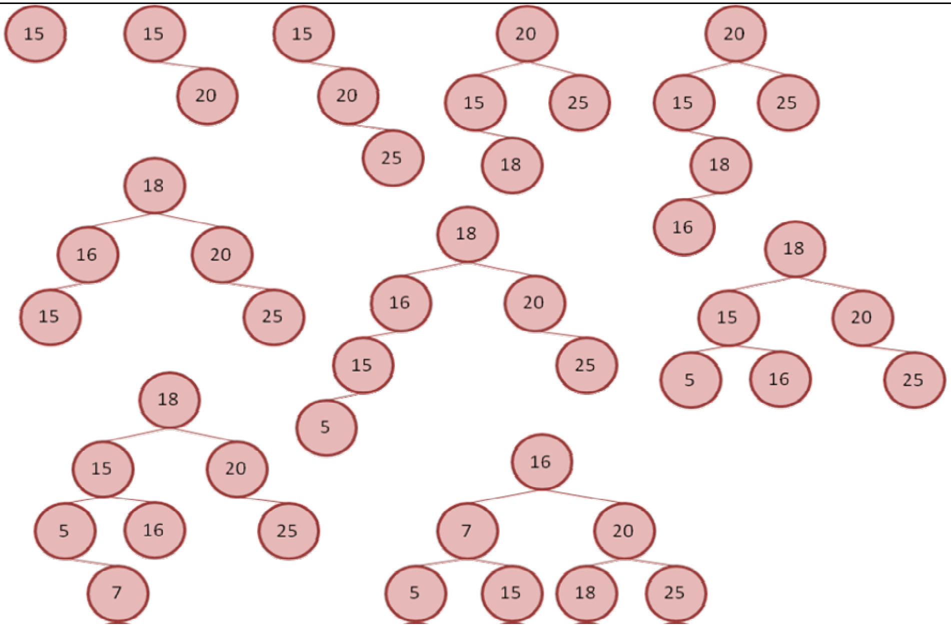
d) Write the pre-order, in-order and post-order traversal of the resulting tree from part c).

ANSWER:



a)

b) Pre-orden: 15-5-7-20-18-16-25
In-orden: 5-7-15-16-18-20-25
Post-orden: 7-5-16-18-25-20-15



c)
d) Pre-orden: 16-7-5-15-20-18-25
   In-orden: 5-7-15-16-18-20-25
   Post-orden: 5-15-7-18-25-20-16

E-15 y E-16		

A student wants to store his EDA exercises but he is not resolving them orderly. Because of that he wants to use a binary search tree. Each exercise must store the name (that will be compose by the number of the sheet of the exercise and the number of the exercise, for example, exercise 4 of sheet 1 would be 104), the statement and the answer (supposing that answer is a String). Implement in java a binary search tree that stores the exercises and implements the following methods:

a) Replace an exercise: the student has modified an existing exercise and wants to replace the old one.
b) Include a new exercise.
c) Search an exercise given its name.
d) Delete an exercise.
e) Get the number of exercises that have been solved.

SOLUCIÓN:
```java
package sheet2.e13;

public class Exercise {
	private String name;
	private String enunciado;
	private String solution;
	public Exercise() {
		super();
	}
	public Exercise(String name, String enunciado, String solution)
{
		super();
		this.name = name;
		this.enunciado = enunciado;
		this.solution = solution;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getEnunciado() {
		return enunciado;
	}
	public void setEnunciado(String enunciado) {
		this.enunciado = enunciado;
	}
	public String getSolution() {
		return solution;
	}
	public void setSolution(String solution) {
		this.solution = solution;
	}
	public String toString(){
		return "[ " + name + "\t" + enunciado + "\t" + solution +
" ]";
	}
}

package sheet2.e13;
import sheet2.BinarySearchNode;
import sheet2.BinarySearchTree;
public class ExerciseBST extends BinarySearchTree<Integer,Exercise>{
	public void insertExercise(Exercise e){
		BinarySearchNode<Integer, Exercise> v = new
BinarySearchNode<Integer, Exercise>(new Integer(e.getName()), e);
		this.insertNode(v);
	}
	public Exercise getExercise(String name){
		BinarySearchNode<Integer, Exercise> v =
this.searchNode(this.root(), new Integer(name));
		if(v!=null){
			return v.getValor();
		}
		return null;
	}
	public Exercise replaceExercise(Exercise o, Exercise n) {
		BinarySearchNode<Integer, Exercise> v =
this.searchNode(this.root(), new Integer(o.getName()));
		Exercise result = null;
		if(v!=null){
			result = v.getValor();
			v.setValor(n);
			return result;
		}
		return null;
	}
	public Exercise deleteExercise(String name){
		BinarySearchNode<Integer, Exercise> v =
this.removeNode(new Integer(name));
		if(v!=null){
			return v.getValor();
		}
		return null;
	}
	public void showPreOrder(){
```

```
            BinarySearchNode<Integer, Exercise> aux = this.root();
            while(aux!=null){
                  System.out.println(aux.getValor().toString());
                  aux = this.preorderNext(aux);
            }
            System.out.println("---------------");
      }
      public void showInOrder(){
            BinarySearchNode<Integer, Exercise> aux = this.root();
            while(aux.left()!=null){
                  aux = aux.left();
            }
            while(aux!=null){
                  System.out.println(aux.getValor().toString());
                  aux = this.inorderNext(aux);
            }
            System.out.println("---------------");
      }
      public void showPostOrder(){
            BinarySearchNode<Integer, Exercise> aux = this.root();
            while(aux.left()!=null){
                  aux = aux.left();
            }
            while(aux!=null){
                  System.out.println(aux.getValor().toString());
                  aux = this.postorderNext(aux);
            }
            System.out.println("---------------");
      }
}

package sheet2.e13;
public class Student {
      ExerciseBST exercises = new ExerciseBST();
      /**
       */
      public Exercise replaceExercise(Exercise o, Exercise n){
            if(!o.getName().equals(n.getName())){
                  System.out.println("They are not the same exercise.
Can not replace it.");
                  return null;
            }
            return exercises.replaceExercise(o,n);
      }
      /**
       */
      public void addExercise(Exercise n){
            exercises.insertExercise(n);
      }
      /**
       */
      public Exercise deleteExercise(String name){
            return exercises.deleteExercise(name);
      }
      /**
       */
      public Exercise searchExercise(String name){
            return exercises.getExercise(name);
      }
      /**
       */
```

```
	public int listNumberExercises(){
		return exercises.size();
	}
	public void printExercises(){
		exercises.showInOrder();
	}
	public static void main (String args[]){
		Student s1 = new Student();
		Exercise e1 = new Exercise("101", "enun101", "sol101");
		Exercise e2 = new Exercise("107", "enun107", "sol107");
		Exercise e3 = new Exercise("112", "enun112", "sol112");
		Exercise e4 = new Exercise("205", "enun205", "sol205");
		Exercise e5 = new Exercise("220", "enun220", "sol220");
		Exercise e6 = new Exercise("206", "enun206", "sol206");
		s1.addExercise(e3);
		s1.addExercise(e6);
		s1.addExercise(e2);
		s1.addExercise(e1);
		s1.addExercise(e5);
		s1.addExercise(e4);
		s1.printExercises();
		Exercise ee = s1.replaceExercise(e2, new
Exercise("107","enun107_2","sol107_2"));
		if(ee!=null){
			s1.printExercises();
		}
		Exercise ee2 = s1.replaceExercise(e3, new
Exercise("113","enun113_2","sol113_2"));
		if(ee2!=null){
			s1.printExercises();
		}
		s1.deleteExercise("206");
		s1.printExercises();
	}
}
```

E-17		

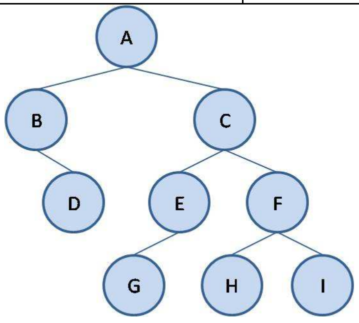


Figure E-20-1

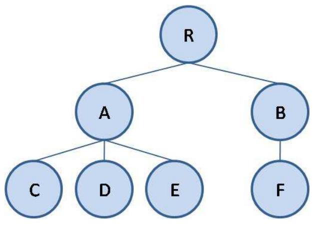


Figure E-20-2

a) Draw the resulting binary tree of the following sequence supposing that the sequence for figure E-20-1 is AB/D//CEG///FH//I// where every / represents a null node.

ABD//E//C/F//

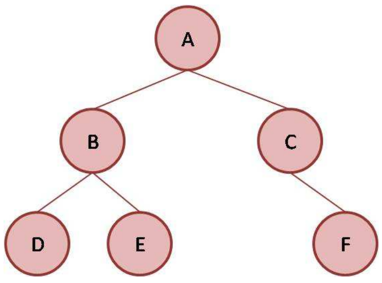
b) Draw the resulting binary tree of the following sequence supposing that the sequence for figure E-20-1 is A'B'/DC'E'G/F'HI where each / represents a null node.

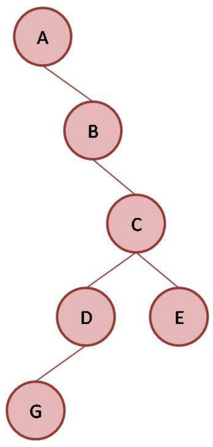
A'/B'/C'D'G/E

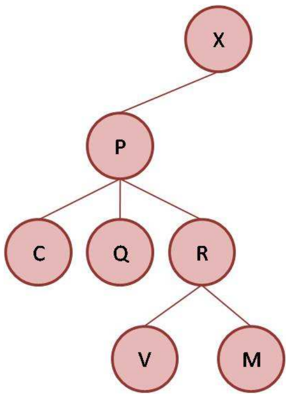
c) Draw the resulting GENERAL tree of the following sequence supposing that the sequence for figure E-20-2 is RAC)D)E))BF))) where each / represents a null node.

XPC)Q)RV)M))))

ANSWER:

a) 

b) 

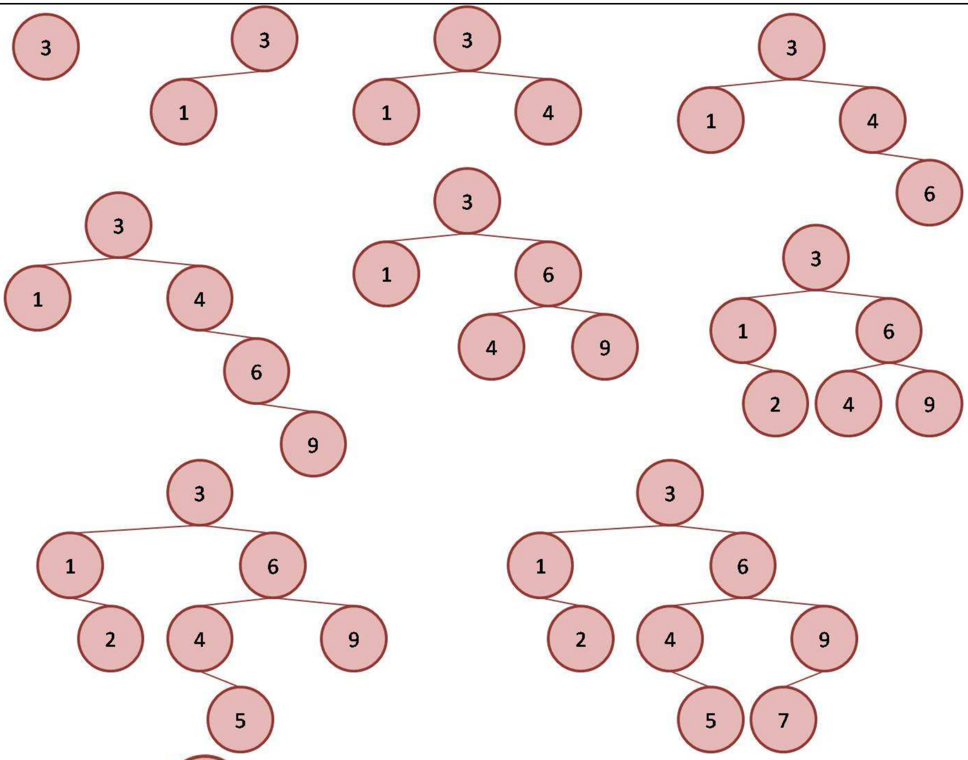
c) 

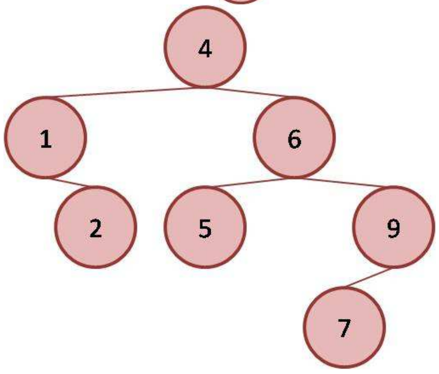
E-18		

a) Draw the resulting and intermediate trees of inserting 3, 1, 4, 6, 9, 2, 5, 7 in a completely balanced BST initially empty.

b) Show the result of deleting the root of the tree.

ANSWER:


a)

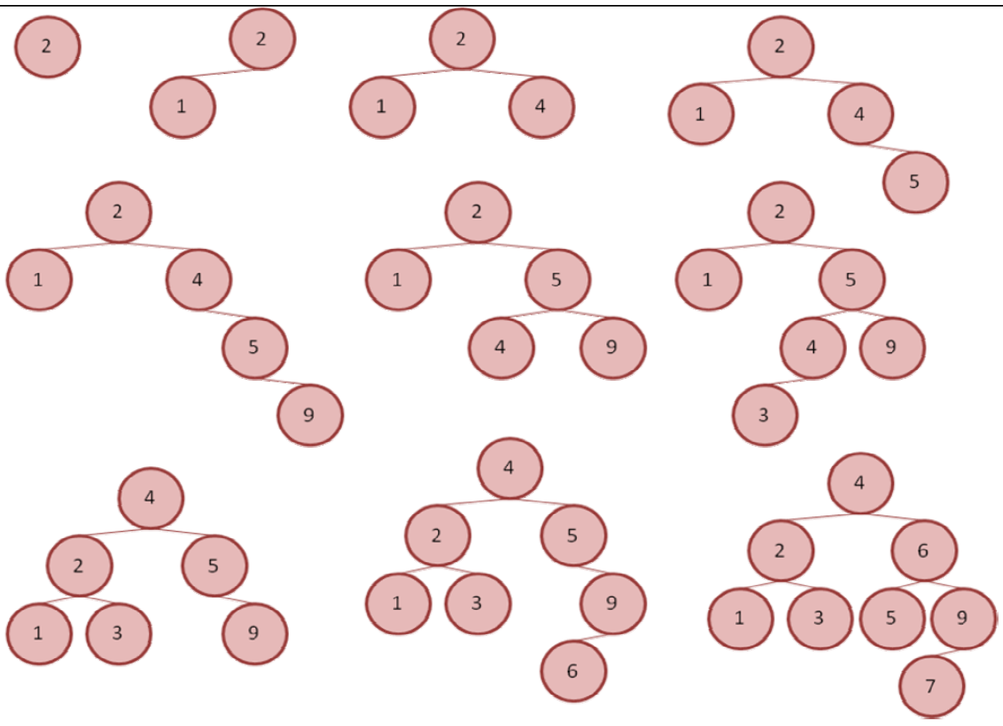

b)

E-19		

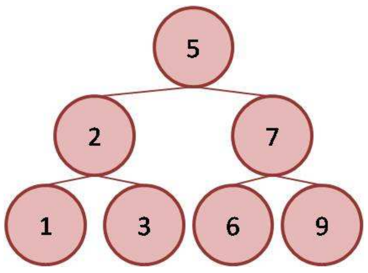
a) Draw the resulting and intermediate trees of inserting 2, 1, 4, 5, 9, 3, 6, 7 in an AVL tree initially empty.

b) Show the result of deleting the root of the tree.

ANSWER:



a)
b) The exercise can be resolved in two ways.



E-20		

Describe a method that generates a random binary tree containing n nodes with different key values between the range of 1 and n.

ANSWER 1:

```
Using an auxiliar array to see if the key is still used.

public BinarySearchTree<Integer> generateRandomTree(int n){
      BinarySearchTree<Integer> tree =new BinarySearchTree<Integer>();
      boolean[] libres = new boolean[n];
      for(int i=0;i<libres.length;i++){
            libres[i] = true;
      }
      while(stillFree(libres)){
            int number = (int)(Math.random()*n)+1;
            while(!libres[number-1]){
                  number = (int)(Math.random()*n)+1;
            }
            libres[number-1] = false;
            BinarySearchNode<Integer> node =new
                        BinarySearchNode<Integer>(number, number);
            tree.insertNode(node);
      }
      return tree;
}
private boolean stillFree(boolean[] in){
      for(int i=0;i<in.length;i++){
            if(in[i]==true){
                  return true;
            }
      }
      return false;
}
```

ANSWER 2:

Testing if the key is included in the tree.

```
public BinarySearchTree<Integer,Integer> generateRandomTree2(int n){
      BinarySearchTree<Integer,Integer> tree = new
BinarySearchTree<Integer,Integer>();
      while(tree.size<n){
            int number = (int)(Math.random()*n)+1;
            while(tree.searchNode(tree.root, new
Integer(number))!=null){
                  number = (int)(Math.random()*n)+1;
            }
            BinarySearchNode<Integer,Integer> node = new
BinarySearchNode<Integer,Integer>(number, number);
            tree.insertNode(node);
      }
      return tree;
}
```