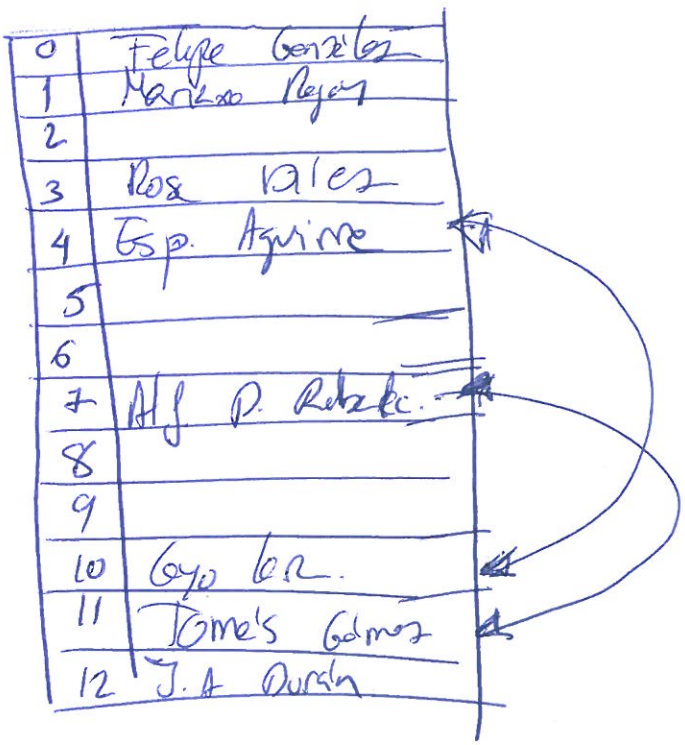
Ejercicio 1.1

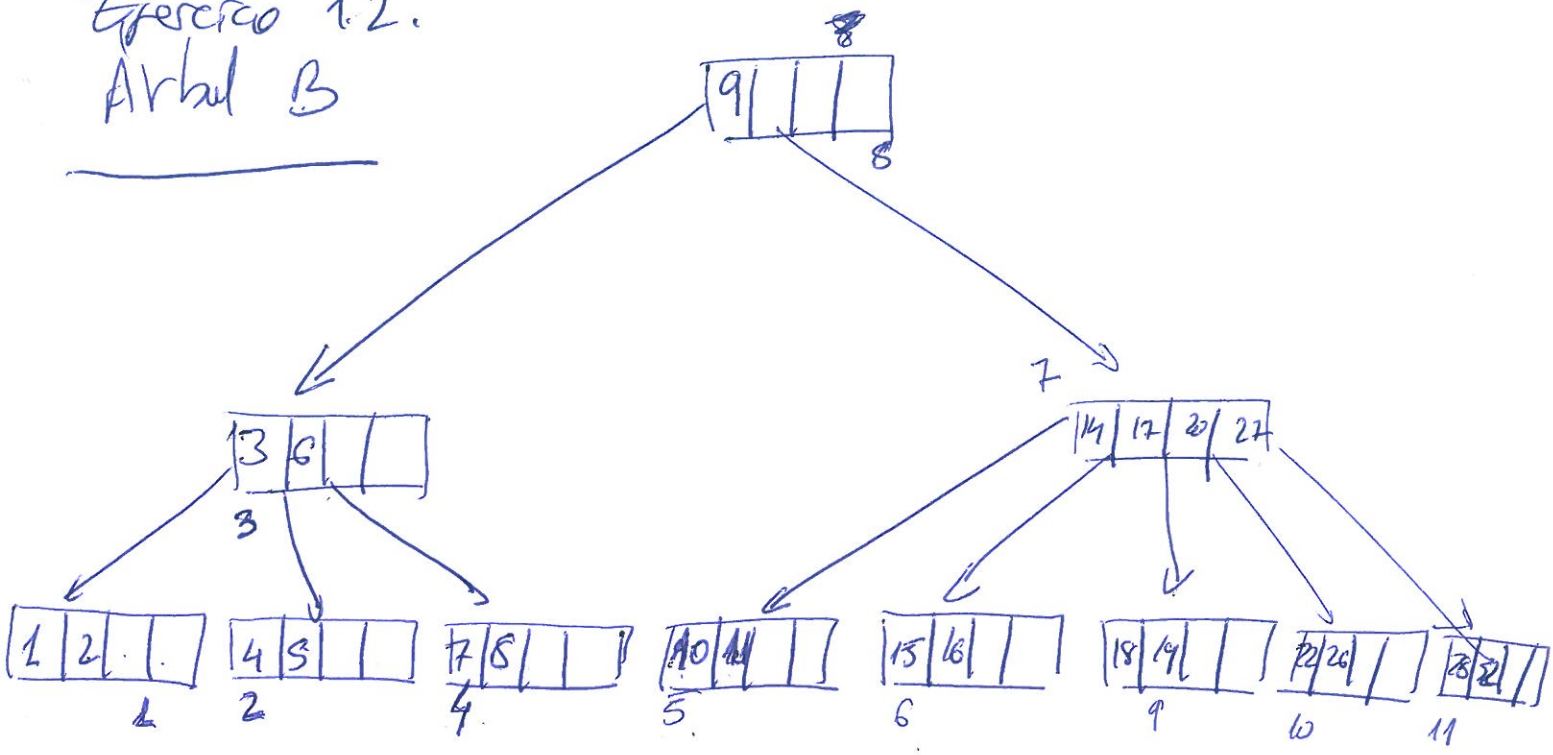
Posiciones en la tabla

	hash	desp	hash + desp mod 13
Rojo	1	0	1
Rubio	7	0	7
Lor	10		10
Durán	12		12
Díez	3		3
Aguirre	10	2 * Lor.	4
Gómez	7	4 * Rubio.	11
González			0



Ejercicio 1.2.

Arbol B

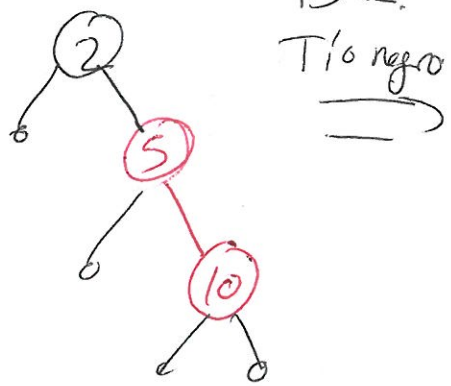


Formato Fichero

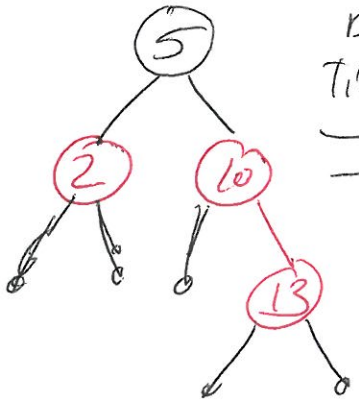
Raiz 8

Nodo	H1	clave1	valor1	H2	clave2	valor2	H3	clave3	valor3	H4	clave4	valor4	H5	Padre
0														
1	-1	[illegible]	Abrahm	-1	2	Raul	-1	-1	-1	-1	-1	-1	-1	3
2	-1	4	Adgel	-1	[illegible]	[illegible]	-1	-1	-1	-1	-1	-1	-1	3
3	1	3	Vela	2	6	Segue	4	-1	-1	-1	-1	-1	-1	8
4	-1	7	Roberts	-1	8	Nora	-1	-1	-1	-1	-1	-1	-1	3
5	-1	10	[illegible]	-1	11	[illegible]	-1	-1	-1	-1	-1	-1	-1	7
6	-1	15	Crews	-1	16	Hernández	-1	-1	-1	-1	-1	-1	-1	7
7	5	14	Adrian	6	17	Rocio	9	20	Lundgren	[illegible]	27	Williams	11	8
8	3	9	Yu	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	-1	18	[illegible]	-1	19	Li	-1	-1	-1	-1	-1	-1	-1	7
10	-1	22	Morris	-1	24	VanQuinne	-1	-1	-1	-1	-1	-1	-1	7
11	-1	28	Stellone	-1	32	Arnold	-1	-1	-1	-1	-1	-1	-1	7
12														

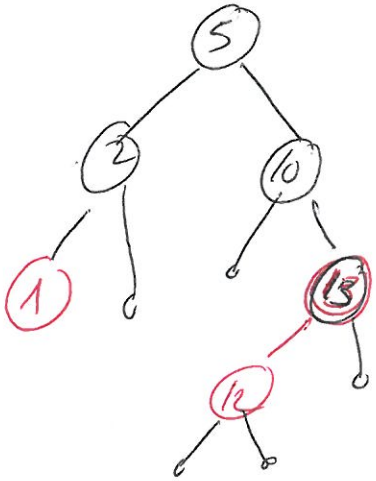
Ejercicio 1.3. Árbol RN



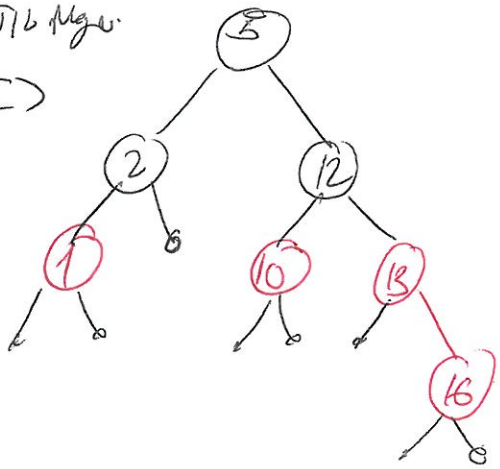
DR.
Tío negro



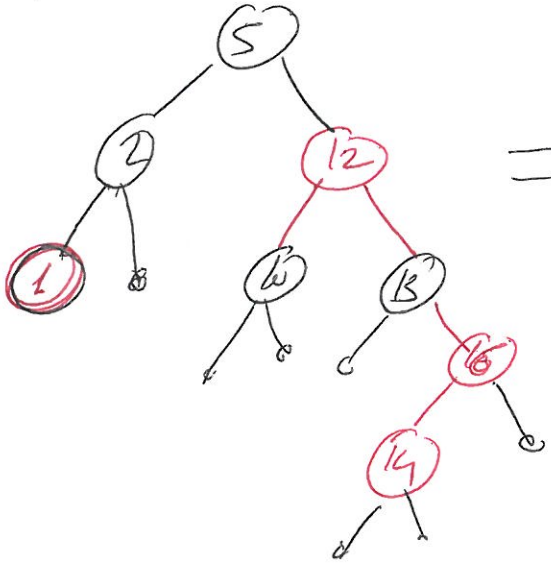
DR.
Tío roja



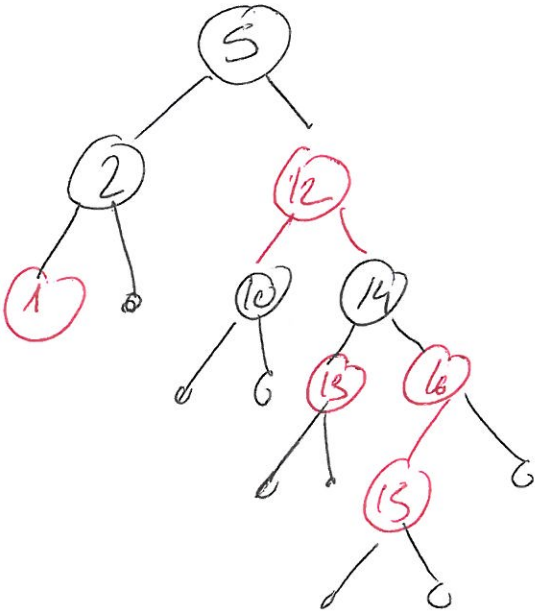
DR
Tío Negro



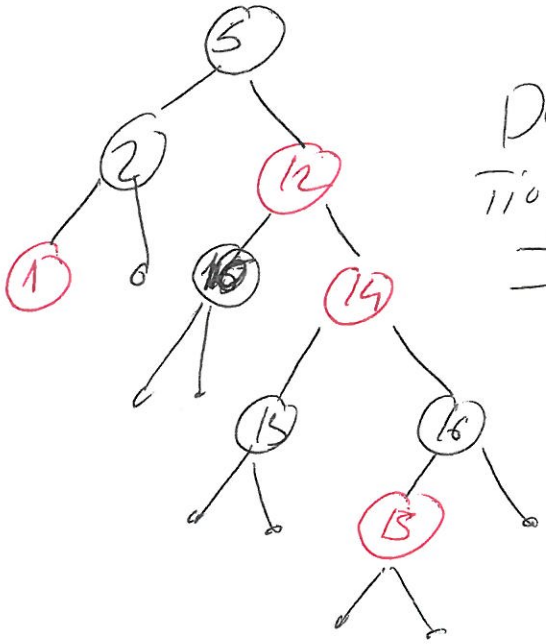
DR
Tío roja



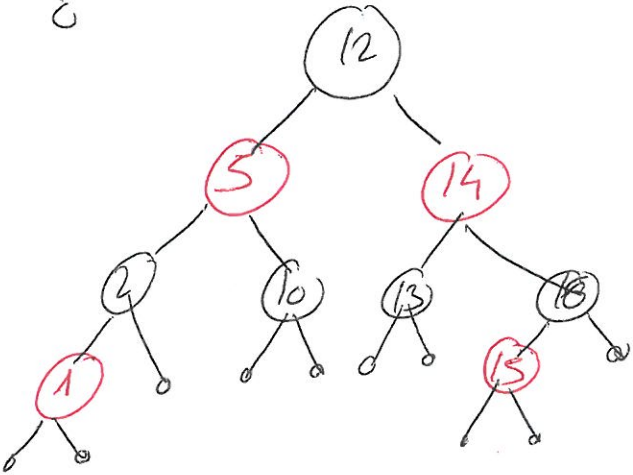
DR
Tío roja



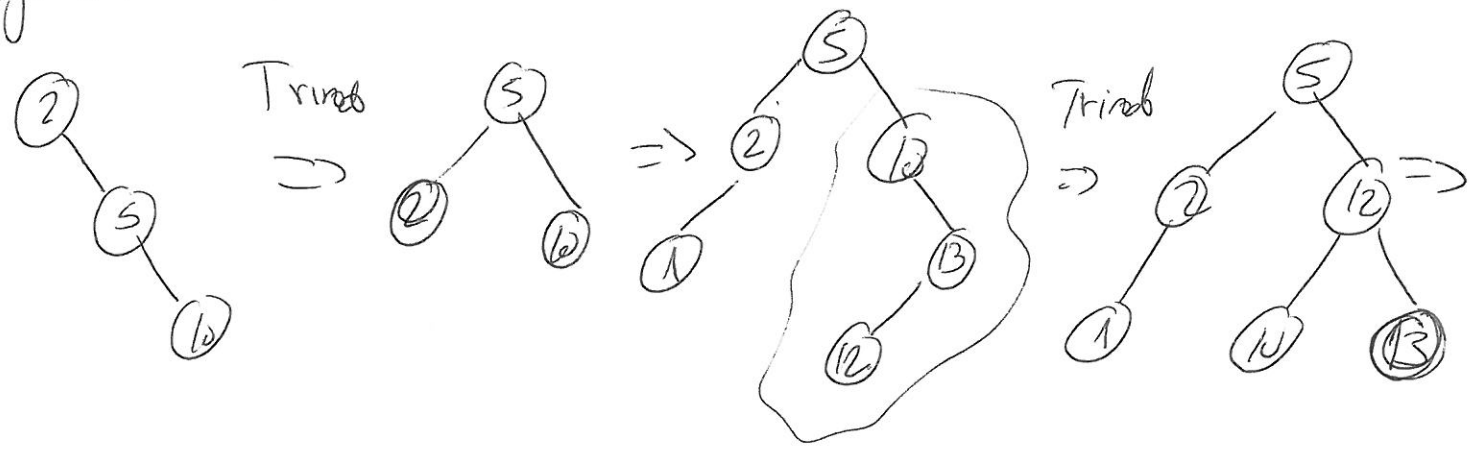
DR
Tío Roja

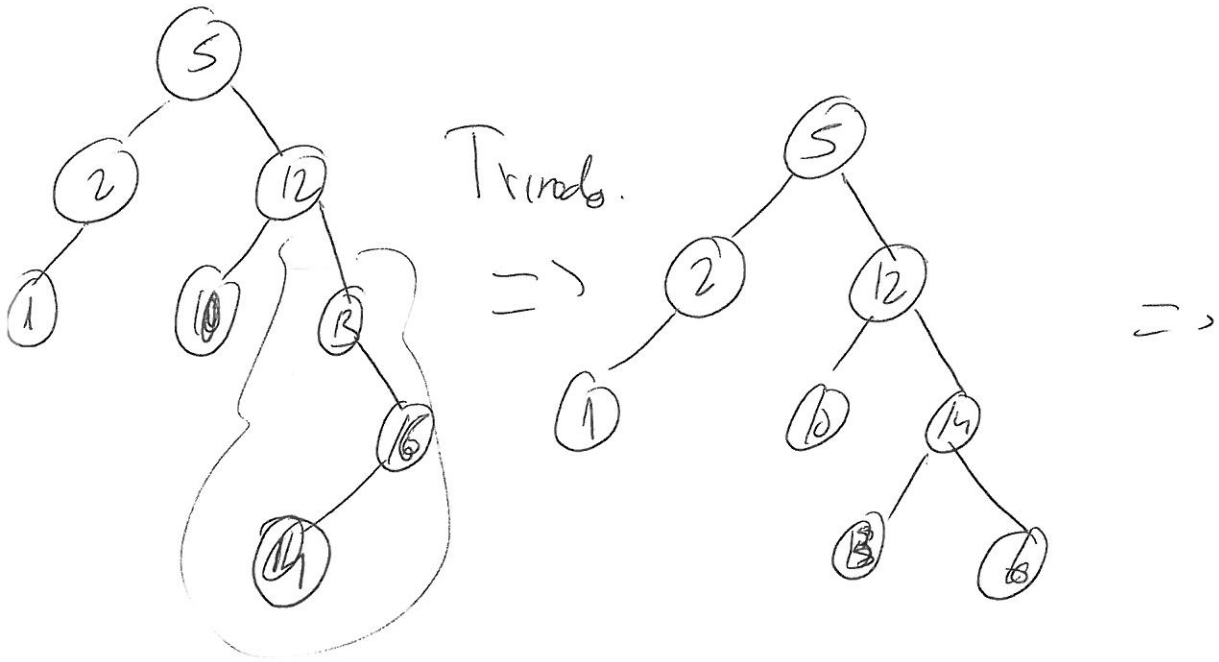


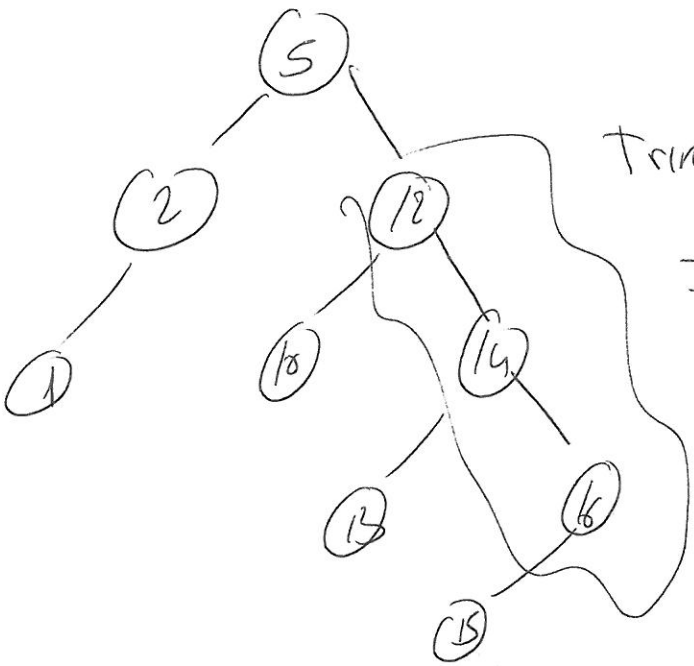
DR
Tío roja



Ejercicio 1.3 Árbol AVL







Ejercicio 2.a

```
    private boolean recursiveEqual(TreeNode<E> n1, TreeNode<E> n2){
       if(n1.element().equals(n2.element())){
              List<TreeNode<E>> h1 = n1.getChildren();
              List<TreeNode<E>> h2 = n2.getChildren();
              // Para que funcione correctamente debe redefinirse
              // el método equals() dentro de la clase TreeNode<E>
              if(h1.containsAll(h2) && h2.containsAll(h1)){
                     for(TreeNode<E> n : h1){
                            int i = h2.indexOf(n);
                            if(this.recursiveEqual(n, h2.get(i)))´
                                   return false;
                            }
                     }
              }
              else{
                     return false;
              }
       }
       else{
              return false;
       }
       return true;
    }

    public boolean equalTree(LinkedTree<E> t){
       boolean result = false;
       if(this.size() == t.size()){
              TreeNode<E> n1 = (TreeNode<E>)this.root();
              TreeNode<E> n2 = (TreeNode<E>)t.root();
              result = this.recursiveEqual(n1, n2);
       }
       return result;
    }
```

Ejercicio 2.b

```
// Si queremos clonar también la info tipo E hay que modificar la cabecera
// de la clase TreeNode<E extends Clonable> y así usar el método clone()
   private void subTreeAux(LinkedTree<E> t, TreeNode<E> node,
List<TreeNode<E>> children){

       for(TreeNode<E> child : children){
              E info = child.element();
              t.add(info, node);
              this.subTreeAux(t, child, child.getChildren());
       }
    }

    public LinkedTree<E> subTree(Position<E> p){
       LinkedTree<E> t = new LinkedTree<E>();
       TreeNode<E >node = this.checkPosition(p);
       E info = node.element();
       t.addRoot(info);
       subTreeAux(t, node, node.getChildren());

       return t;
    }
```

Ejercicio 3.a

```
public class GestorViajes {
     private Map<String, List<Viaje>> ciudadesOrigen;
     private Map<String, List<Viaje>> ciudadesDestino;
     private List<Viaje> viajes;
     private Set<String> ciudades;
```

Ejercicio 3.b

```
     public GestorViajes(){
       this.ciudadesOrigen = new HashTableMap<String, List<Viaje>>();
       this.ciudadesDestino = new HashTableMap<String, List<Viaje>>();
       this.viajes = new ArrayList<Viaje>();
       this.ciudades = new HashSet<String>();
     }
     public void addViaje(Viaje viaje) {
       viajes.add(viaje);
       ciudades.add(viaje.getCiudadOrigen());
       ciudades.add(viaje.getCiudadDestino());

       addCiudadesOrigen(viaje.getCiudadOrigen(), viaje);
       addCiudadesDestino(viaje.getCiudadDestino(), viaje);
     }

     private void addCiudadesOrigen(String ciudadOrigen, Viaje viaje) {
       List<Viaje> entrada = ciudadesOrigen.containsKey(ciudadOrigen);
       if (!entrada.isEmpty()) {
              entrada.add(viaje);
       } else {
              List<Viaje> nuevaCiudad = new ArrayList<Viaje>();
              nuevaCiudad.add(viaje);
              ciudadesOrigen.put(ciudadOrigen, nuevaCiudad);
       }
     }

     private void addCiudadesDestino(String ciudadDestino, Viaje viaje) {
       List<Viaje> entrada = ciudadesOrigen.containsKey(ciudadOrigen);
       if (!entrada.isEmpty()) {
              entrada.add(viaje);
       } else {
              List<Viaje> nuevaCiudad = new ArrayList<Viaje>();
              nuevaCiudad.add(viaje);
              ciudadesDestino.put(ciudadDestino, nuevaCiudad);
       }
     }
```

Ejercicio 3.c

```
// Ojo que la colección de ciudades puede ser null
     public Iterable<Viaje> getOrigenes(String ciudadOrigen) {
       return ciudadesOrigen.get(ciudadOrigen);
     }

     public Iterable<Viaje> getDestinos(String ciudadDestino) {
       return ciudadesDestino.get(ciudadDestino);
     }
```

Ejercicio 3.d y 3.e

```
    public Iterable<Viaje> getViajes() {
      return viajes;
    }

    public Iterable<String> getCiudades() {
      return ciudades;
    }
}
```