Ariketa guztien ebazpenerako, zuhaitz bitar baten inplemetazio bat ematen digute BinTree klasearen bitartez. Klase honen espezifikazioa hurrengoa da:

```
public class BinTree<T> {
  public BTNode<T> root; // zuhaitzaren erroa

 }
```

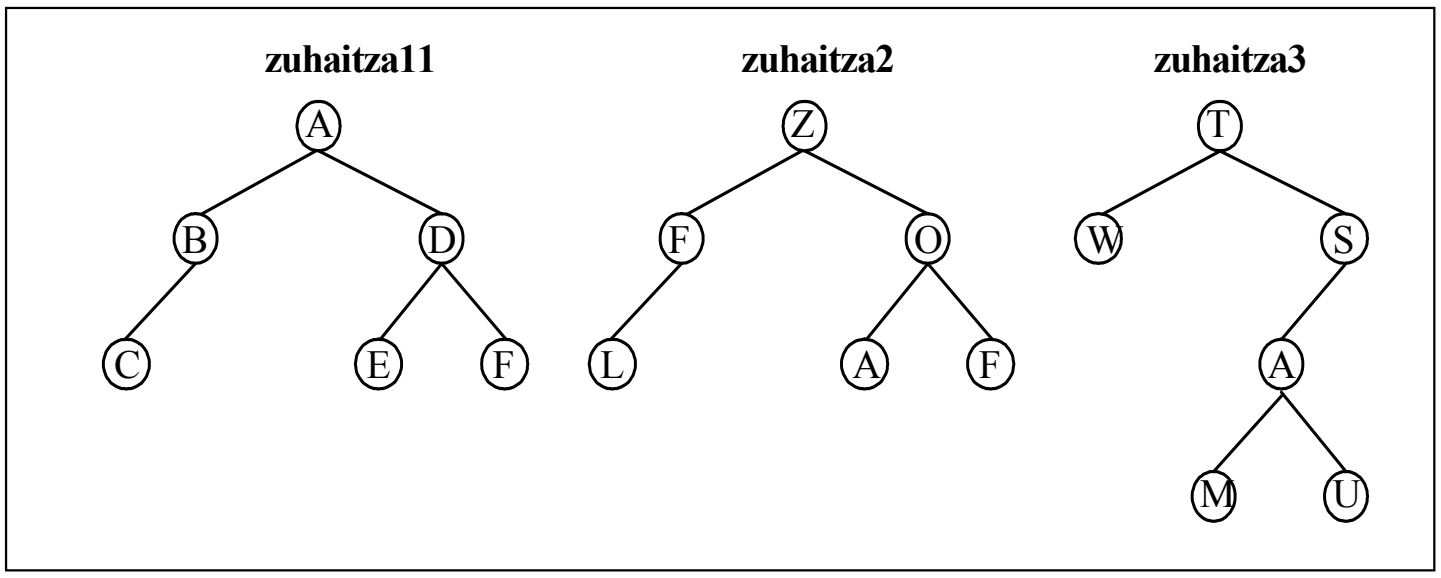
BTNode klasearen espezifikazioa hurrengoa da:

```
public class BTNode<T> {

   T content;
   BTNode<T> left;
   BTNode<T> right;

   //Node bat sortzen da content edukinarekin
   public BTNode(T balio)
```

Eskatzen diren metodo guztiak, BinTree klasearen barnean inplementatu behar dira

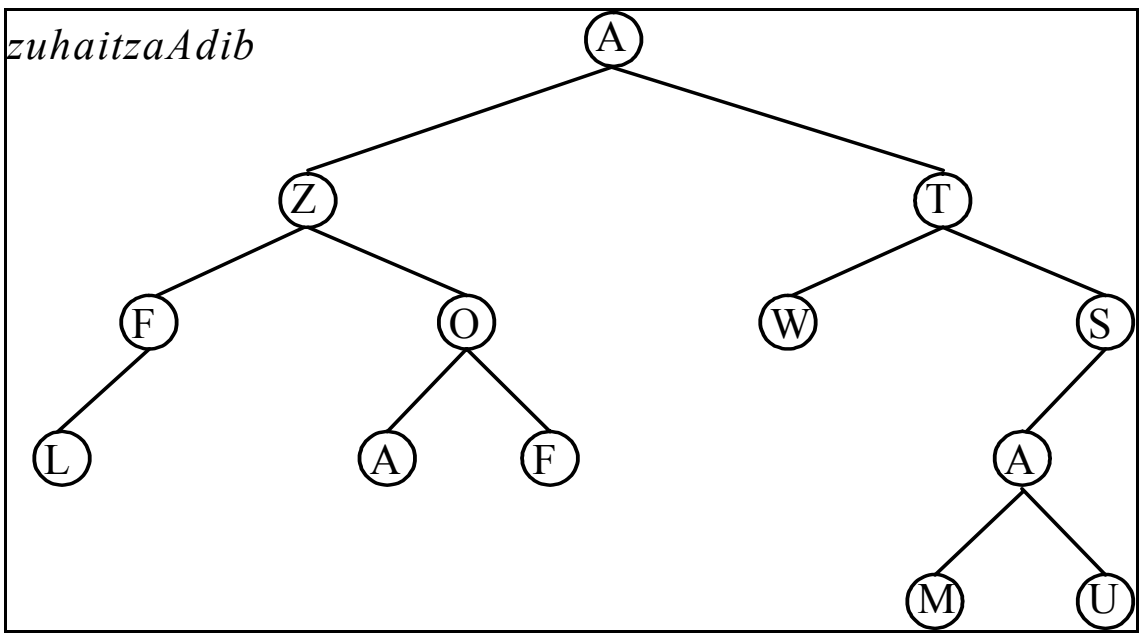
(E94) Diseina eta inplementatu Java-n *egituraBerdina* metodoa. Metodo honek bi zuhaitz bitar jasotzen ditu parametro bezala eta hurrengo emaitza itzultzen du:
true, bi zuhaitzen egitura berdina bada, hau da, zuhaitz berdinak dira beren adabegien balioak izan ezik, eta *false*, beste kasuan

Irudiko abibidean, *egituraBerdina* (zuhaitza1, zuhaitza2) true itzuliko du eta *egituraBerdina* (zuhaitza1, zuhaitza3) false.



(I94) Diseina eta inplementatu Java-n zuhaitz baten mailan dauden elementu kopurua itzultzen duen *nodoKop* metodoa.
Irudian adibidez:

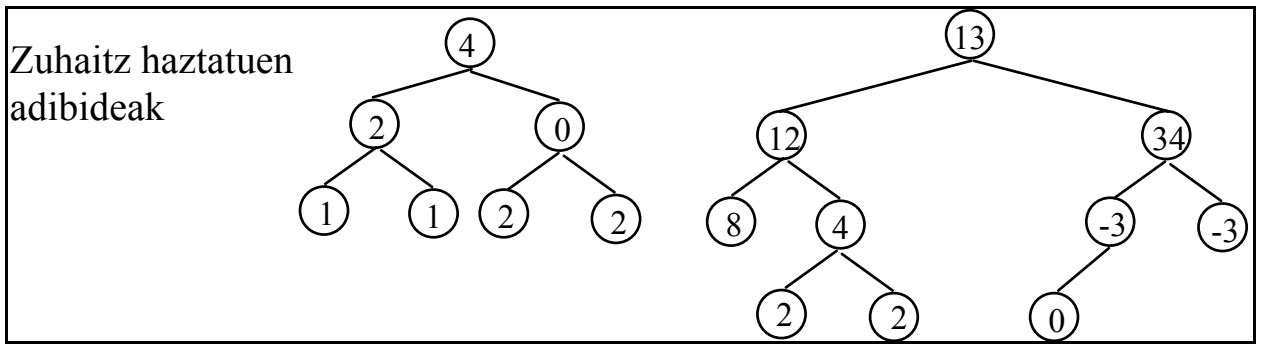
- *zuhaitzaAdib.nodoKop (1)* 1 itzuliko du
- *zuhaitzaAdib.nodoKop(3)* 4 itzuliko du
- *zuhaitzaAdib.nodoKop(5)* 2 itzuliko du



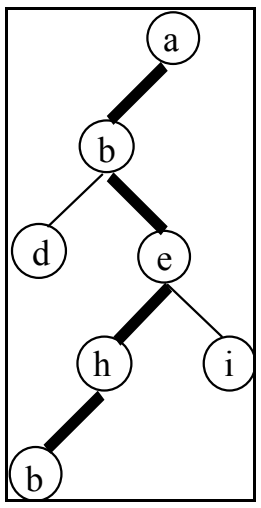
(E95) Diseina eta inplementatu Java-n *isHaztatua* metodoa. Metodo honek zuhaitz bat haztatua den a la ez itzultzen du:

- Zuhaitz bat haztatua da, zuhaitzaren edozein adabegirentzat, ezker azpizuhaitzaren *pisua* eta eskubiko zuhaitzaren *pisuak* berdinak direnean.
- Zuhaitz hutsa, haztatua da.

Zuhaitz baten pisua, adabegi guztien baturari deitzen zaio

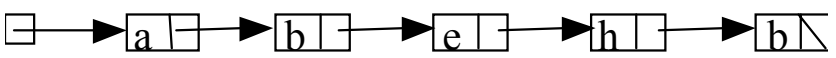


(I95) Diseina eta inplementatu Java-n *isAdarra(LinkedList zerrenda)* metodoa BinTree klasean. Metodoak, *true* itzuliko du, emandako zerrenda zuhaitzaren ***adar oso batekin parekatzen bada***, eta *false*, kontrako kasuan**.**

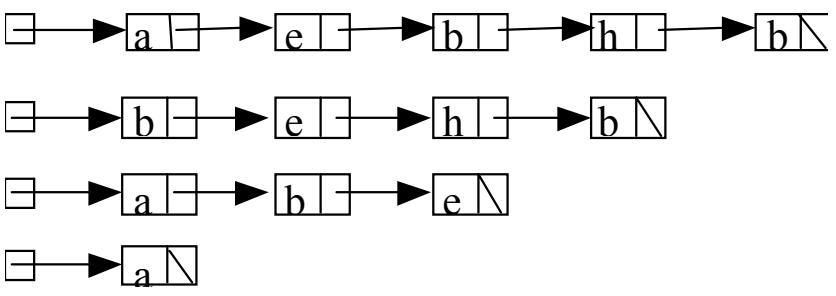


Zuhaitz baten adar bat ***osoa*** da, errotik hasita hosto batean bukatzen bada.

Hurrengo zerrenda, BAI parekatzen da zuhaitzaren adar oso batekin.



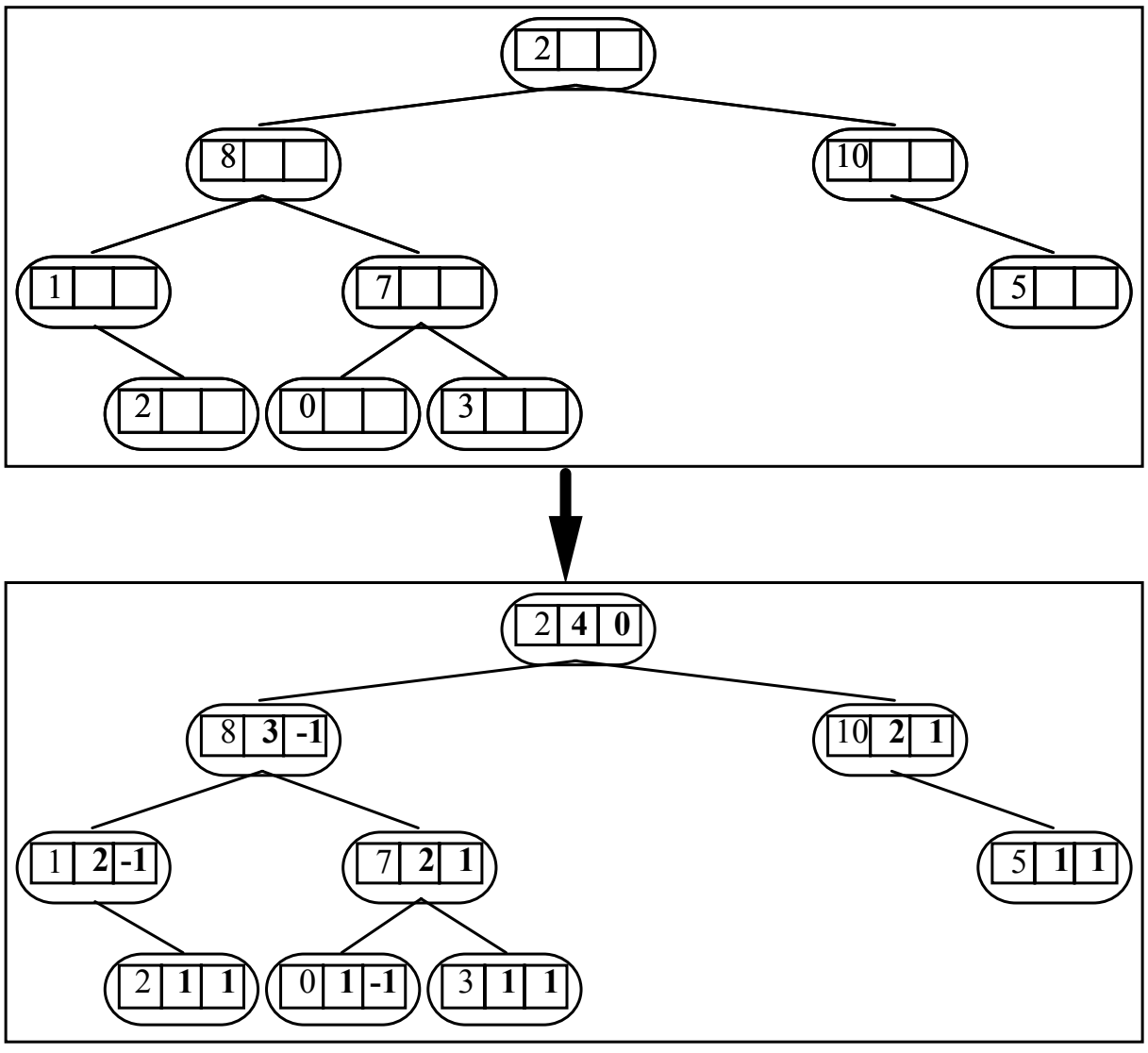
Hurrengo zerrendak, EZ dira parekatzen zuhaitzaren adar oso batekin



(E96)

```
public class Edukia {
     int datu;
     int sakonera;
     int semeMota;
}
```

Node datu motako zuhaitz bitar bat emanda, diseina eta inplementatu zuhaitzaren adabegi guztiak etiketatzen duen metodo bat. **sakonera** atributuan, adabegi horretatik hasten den azpizuhaitzaren sakonera ipiniko du. **semeMota** atributuan, 0 ipiniko du zuhaitzaren erroa bada, -1 azpizuhaitz ezker baten erroa bada eta 1 azpizuhaitz eskubi baten erroa bada.

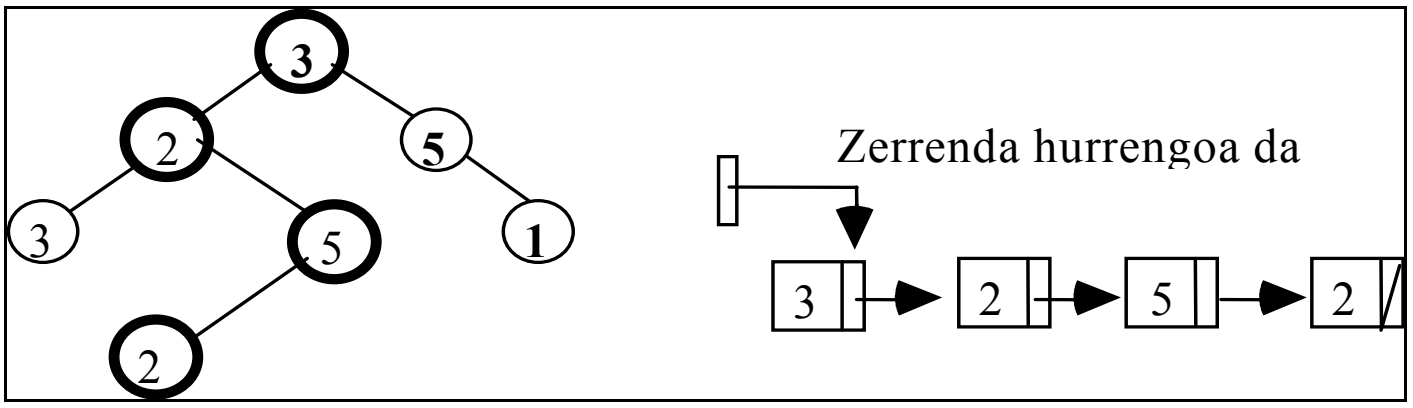


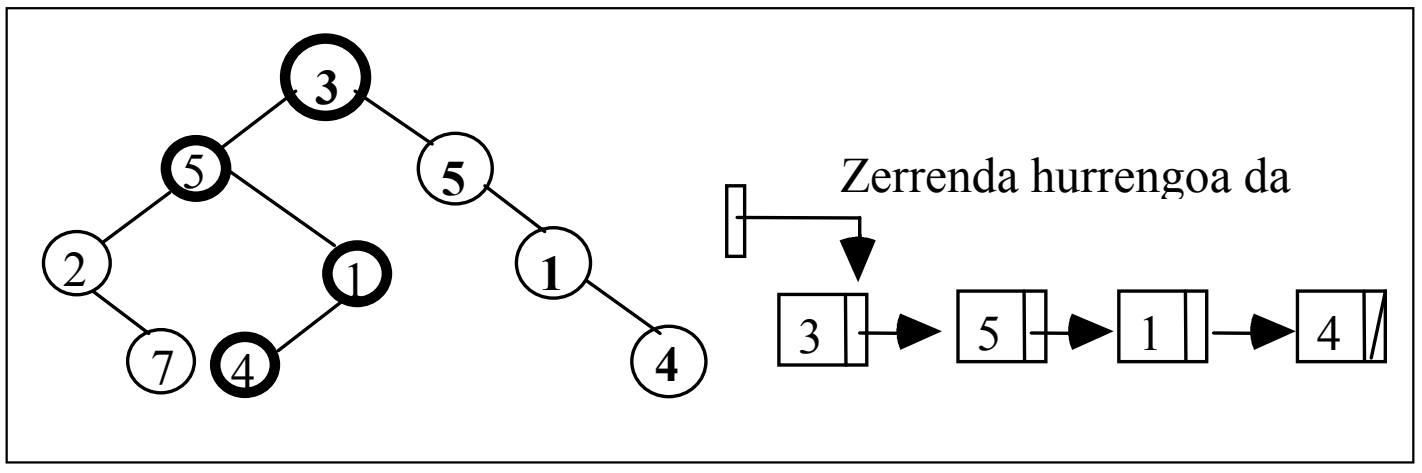
(I96)

Diseina eta inplementau Java-n zuhaitzaren adar luzeagoren adabegi guztien zerrenda bat itzultzen duen metodo bat.

LinkedList adarLuzeena();

Adar bat baino gehiago egongo balitz, zerrendak edozein adarreko adabegiak edukiko ditu (bigarren irudian agertzen den bezala).





(E97)

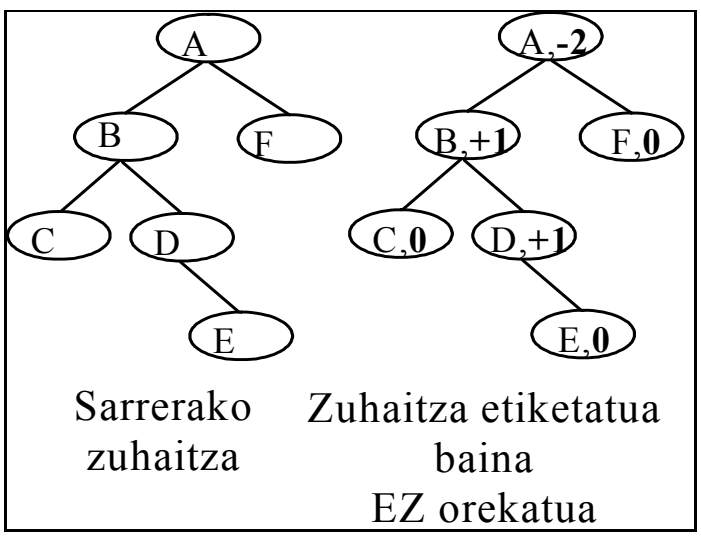
```
public class Edukia {
     Object datu;
     int oreka;
}
```

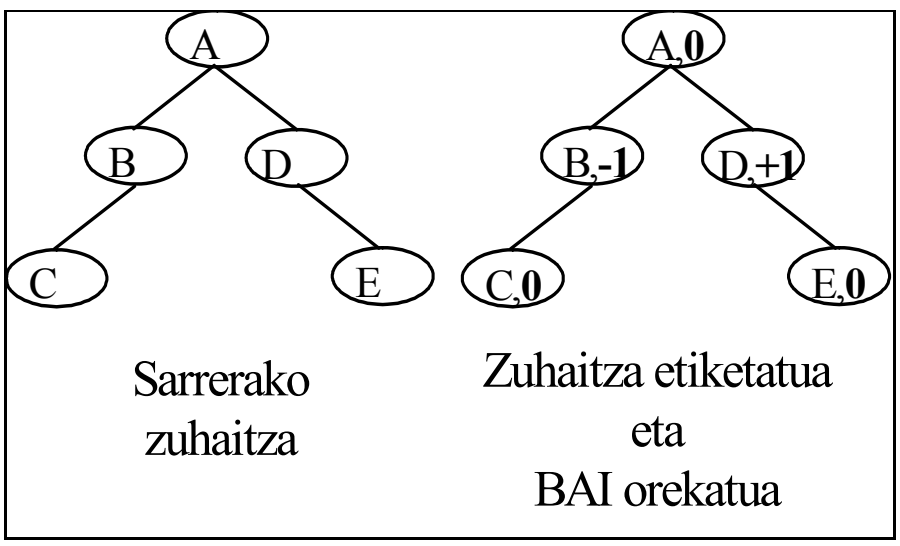
Node datu motako zuhaitz bitar bat emanda, diseina eta inplementatu metodo bat hurrengo portaerarekin:

1. Lehendabizi, adabegi guztien **`oreka`** atributua izendatu hurrengo irizpidearekin:
    - 0, bere azpizuhaitzen sakonerak berdinak badira, edo
    - Beren azpizuhaitzen sakoneren kenketarekin. Zenbakia negatiboa izango da, ezkerreko azpizuhaitzaren sakonera eskubiarena baino handiagoa denean eta positiboa kontrako kasuan.
2. Baita ere, metodoak **balio booleano bat itzuliko du**, *true* zuhaitza orekatua bada edo *false* kontrako kasuan.

Ohar: Zuhaitz bat orerkatua dago, zuhaitzaren adabegi guztientzat, bere azpizuhaitzen sakonera berdina denean edo gehienez diferentzia 1 denean.

Abibideak:



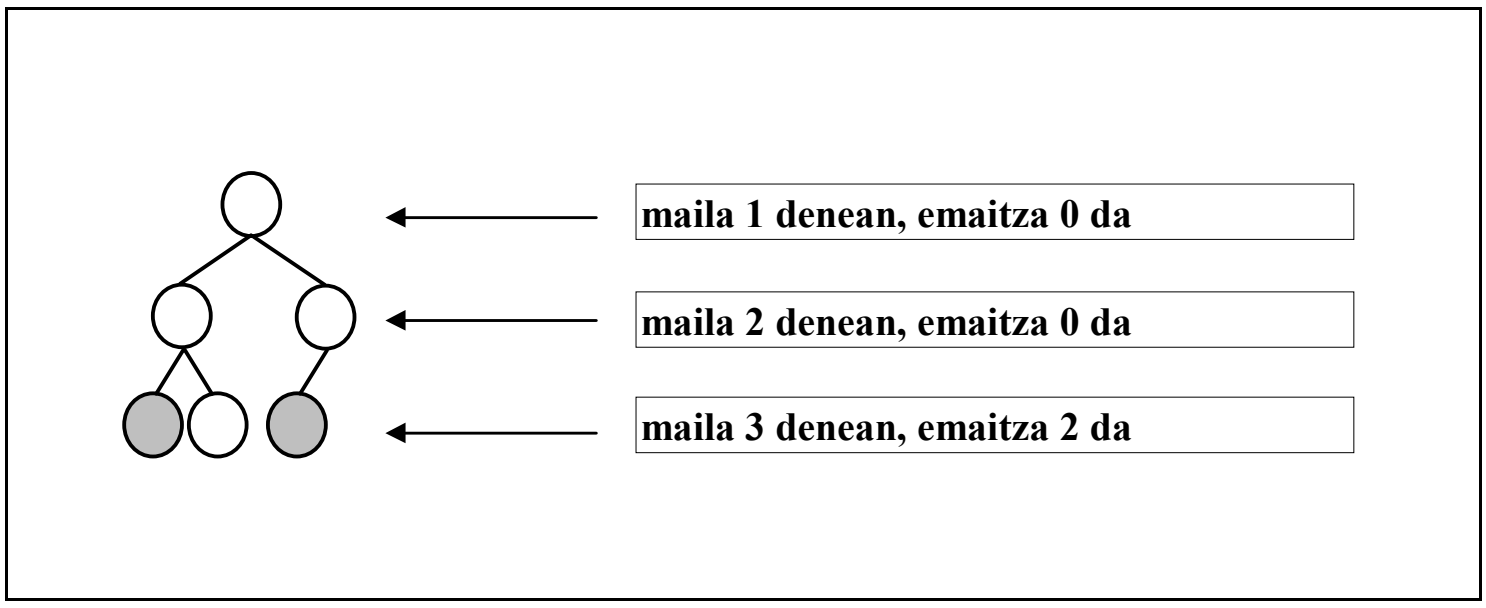


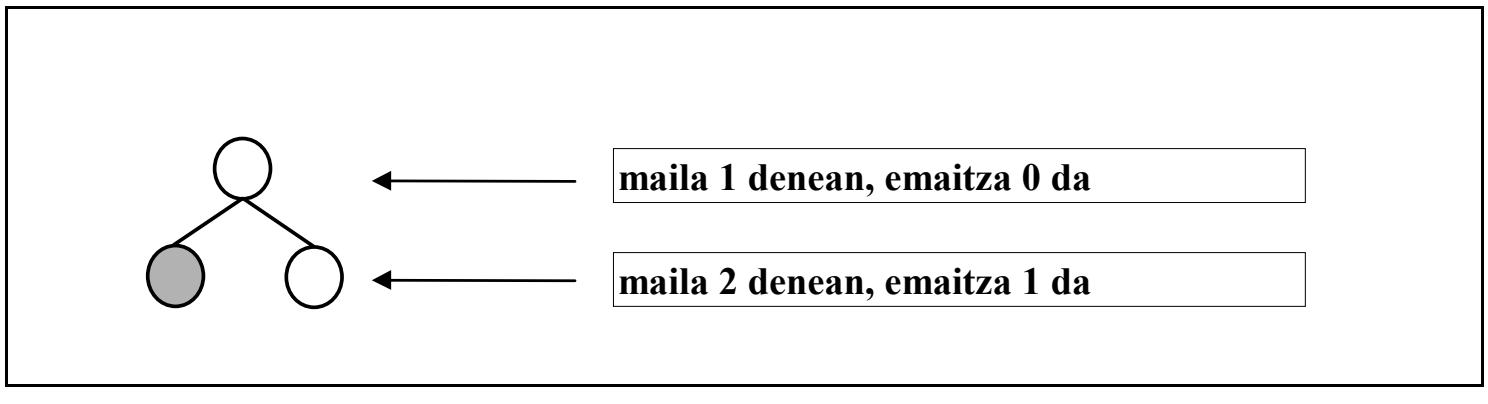
(I97) Zuhaitz bitar bat emanda, diseina eta inplementatu maila batean hurrengo ezaugarriak batera (biak) betetzen dituen adabegi kopuruen metodo bat:

- Hostoa da **eta**
- adabegi ezkerra da.

int hostoaEtaEzkerraKop(int maila)

Adibideak:





(E98)

Zuhaitz bitar baten adabegi bat, bere azpizuhaitz ezkerra eta eskubia hutsa EZ direnean, **2 gradukoa** dela esaten da.

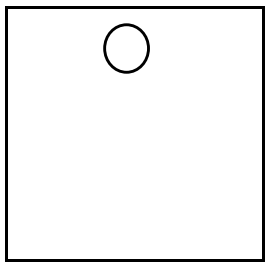
Zuhaitz bitar bat **2-orekatua** dela esaten da hutsa denean, edo aldi berean hurrengo baldintzak betetzen badira:

- Azpizuhaitz ezkerra **2-orekatua** bada.
- Azpizuhaitz eskuina **2-orekatua** bada..
- Ezkerreko eta eskuineko azpizuhaitzen **2 graduko** adabegi kopurua berdina denean.

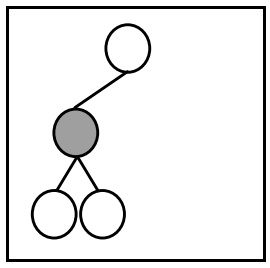
Diseina eta **inplementatu** metodo bat, true itzultzen duena zuhaitza 2-orekatua bada, eta false beste kasuan.

Adibideak:

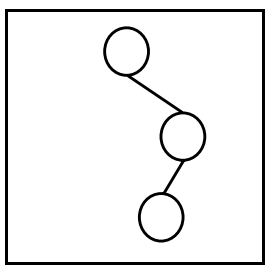
(gradu 0 edo 1eko adabegiak ◯ baten bidez errepresentatzen ditugu eta ● 2 gradukoak)



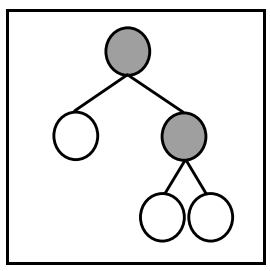
Zuhaitz 2-orekatua
(2 graduko adabegirik gabe)



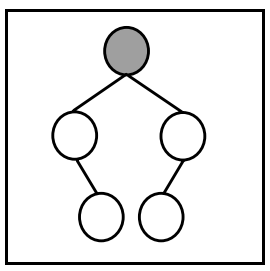
Zuhaitz **ez** 2-orekatua
(adabegi bat 2 gradukoa)



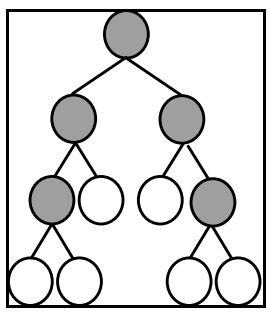
Zuhaitz 2-orekatua
(2 graduko adabegirik gabe)



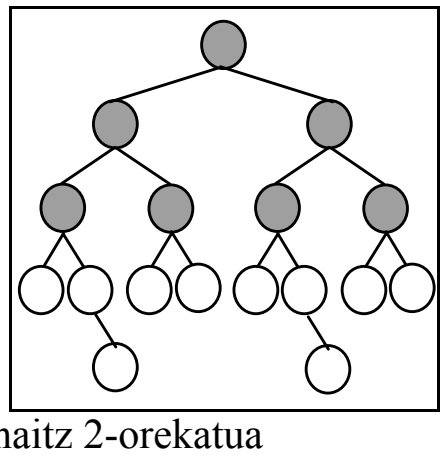
Zuhaitz **ez** 2-orekatua
(2 adabegi 2 gradukoak)



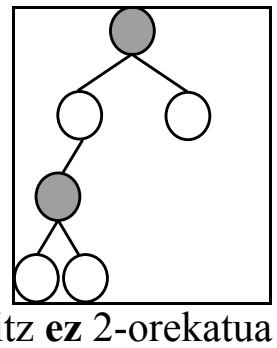
Zuhaitz 2-orekatua
(adabegi bat 2 gradukoa)



Zuhaitz **ez** 2-orekatua
(5 adabegi 2 gradukoak)



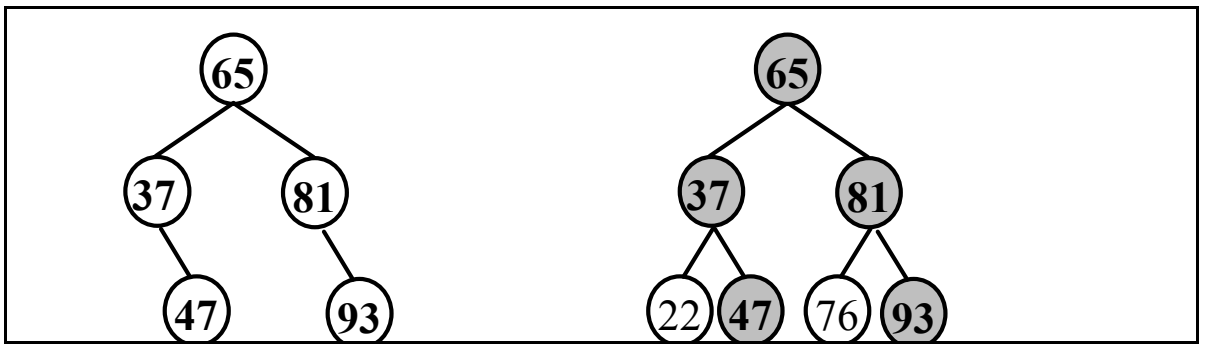
Zuhaitz 2-orekatua
(7 adabegi 2 gradukoak)



Zuhaitz **ez** 2-orekatua
(2 adabegi 2 gradukoak)

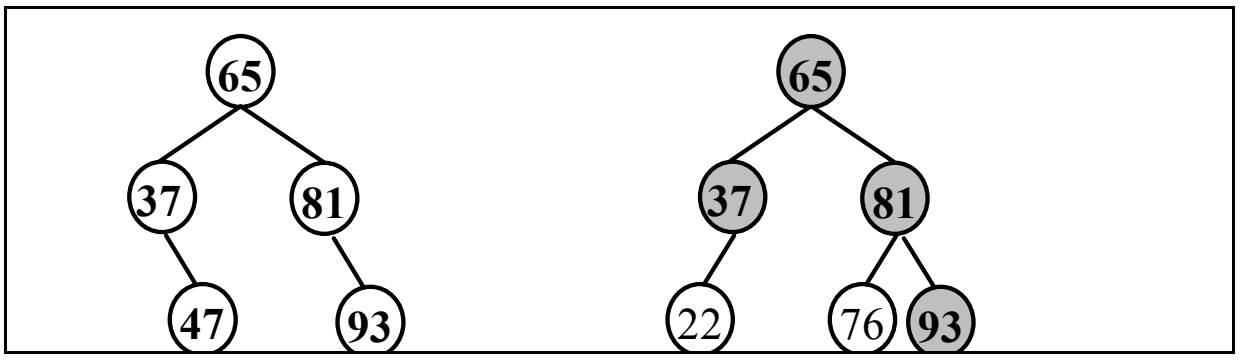
(I98)

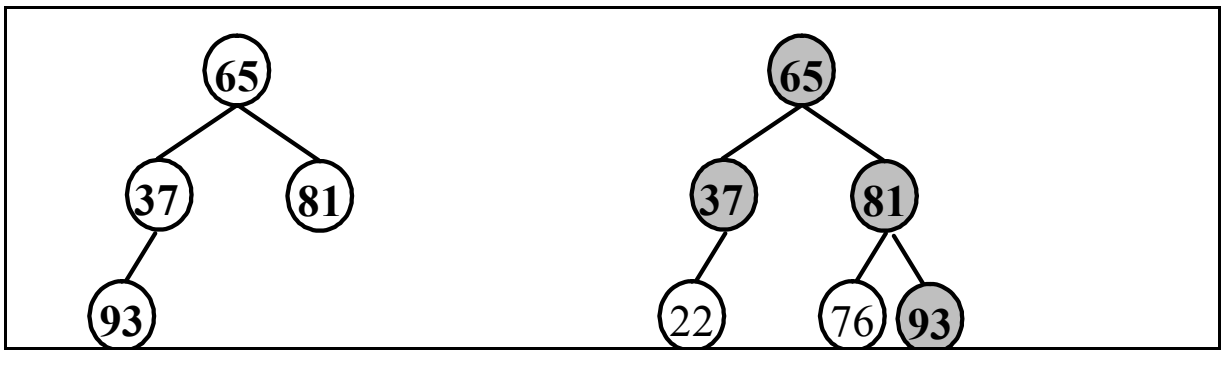
Zuhaitz bitar bat (**zuhaitza1**) beste zuhaitz bitar baten (**zuhaitza2**) aurrizkia dela esaten da, **zuhaitza1, zuhaitza2**ren hasierako parte batekin parekatzen bada, bai elementuen edukinetan baita bere kokapenean ere. Hurrengo irudian, **zuhaitza1**, **zuhaitza2**ren aurrizkia da:



Diseina eta inplementatu **isAurrizkia(BinTree zuhaitza1, BinTree zuhaitza2)** metodoa, true itzultzen duena *zuhaitza1*, *zuhaitza2*-ren aurrizkia bada, eta false beste kasuan.

Hurrengo irudietan **zuhaitza1** EZ DA **zuhaitza2**ren aurrizkia:

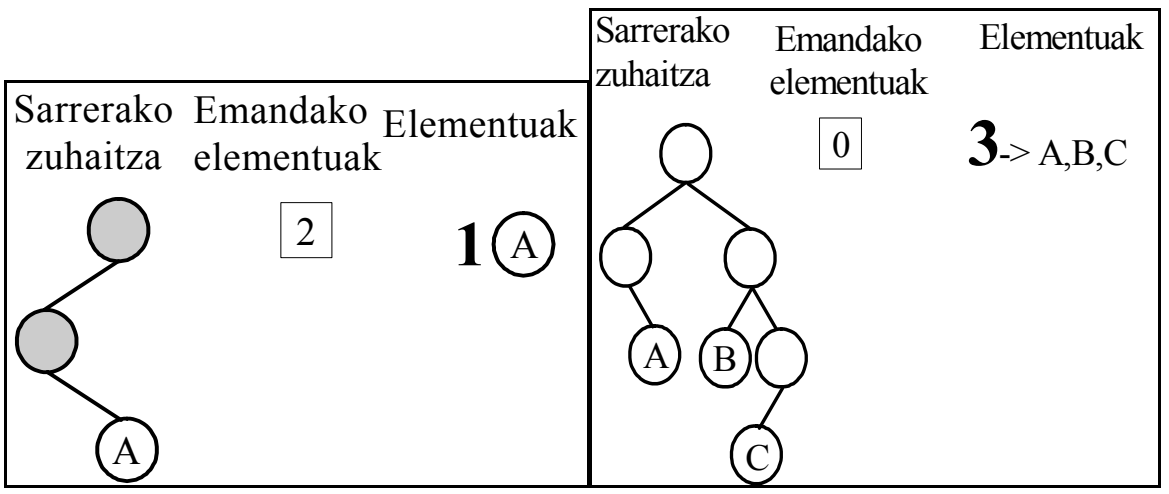




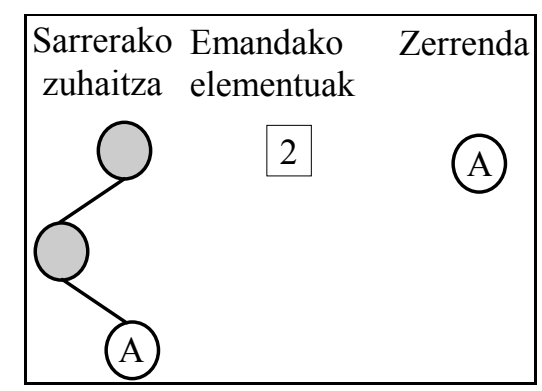
Ariketak

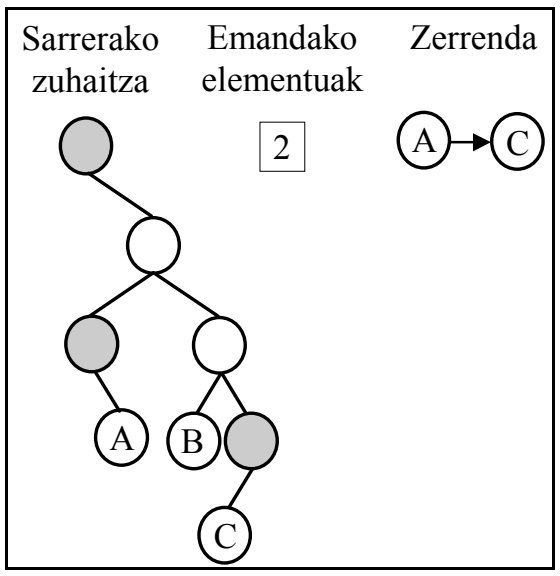
(1) **Diseina** eta **inplementatu int azpizuhaitzHutsBakarra(int abegiKop)** metodoa, hurrengo baldintzak betetzen dituen hosto-adabegien kopuruarekin:

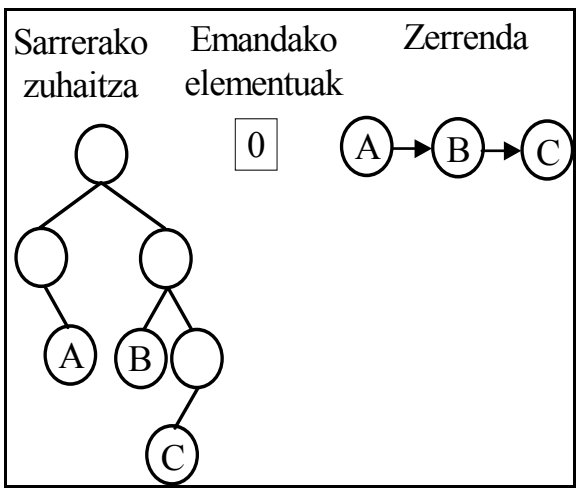
- Adabegia eta erroa lotzen duen adarrean, adabegi kopuru bat dago (emanda datorkigu) azpizuhaitz huts bakar batekin.



(2) **Diseina** eta **inplementatu** aurreko ariketako baldintzak betetzen dituen adabegien zerrenda bat itzulzen duen metodo bat.



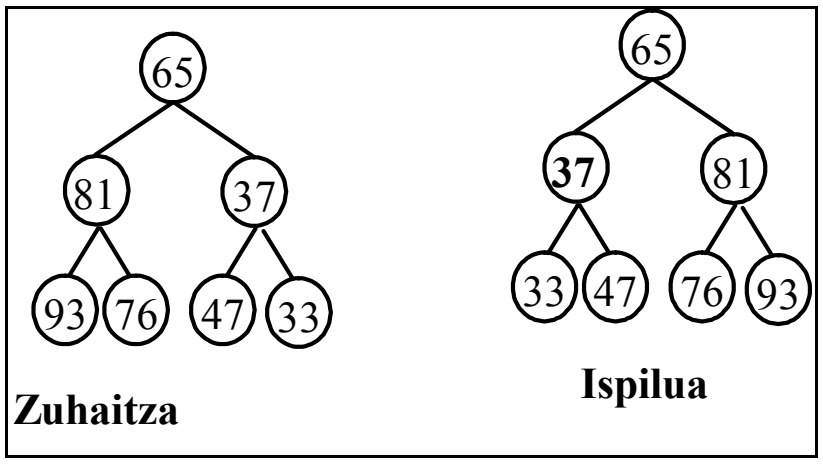




(3) char datu motako zuhaitz bitar bat emanda, maila zenbaki bat eta karaktere bat, **diseina** eta **inplementatu** maila horretan karaktere hori duten adabegien kopurua itzultzen duen metodo bat.

(4) char datu motako zuhaitz bitar bat emanda eta maila zenbaki bat, **diseina** eta **inplementatu** maila horretan dauden karaktereen zerrenda itzultzen duen metodo bat.

(5) Zuhaitz bitar bat emanda**, diseina** eta **inplementatu** bere **ispilu irudia** itzultzen duen metodo bat. Zuhaitz baten ispilua



(6) Bi zuhaitz bitar emanda, **diseina** eta **inplementatu** zuhaitz bat bestearen ispilu irudia erabakitzen duen metodo bat.

(7) Diseina eta **inplemetatu** errepikapenMax() metodoa. Metodo honek elementu oso bat jasoko du, eta zenbaki hori adar batean agertzen den aldi kopuru maximoa itzuliko du.

(8) Diseina eta **inplemetatu** luzeeraMax() metodoa. Metodo honek elementu oso bat jasoko du, eta zenbaki hori agertzen den aldi kopuru maximoaren adarraren luzeera itzuliko du.

(9) Elementu osoko zuhaitz bitar bat emanda, **diseina** eta **inplementatu** pisu gehien duen adarraren luzeera itzultzen duen metodo bat. Adar baten pisua, adar horretan dauden adabegi guztien batura da.

(10) Elementu osoko zuhaitz bitar bat emanda, **diseina** eta **inplementatu** pisu gehien duen adarraren **pisua** itzultzen duen metodo bat. Adar baten pisua, adar horretan dauden adabegi guztien batura da.

(11) Elementu osoko zuhaitz bitar bat emanda, **diseina** eta **inplementatu** pisu gehien duen adarraren **adabegien zerrenda itzultzen** duen metodo bat.

(12) Elementu osoko zuhaitz bitar bat emanda, **diseina** eta **inplementatu** bilaketazko zuhaitz bitarra erabakitzen duen metodo bat. *public boolean isBzb();*

(13) Zuhaitz bitar bat emanda, **diseina** eta **inplementatu** zuhaitzaren sakonera kalkulazen duen metodo bat.

Zuhaitz bitarren ebazpenak

(E94)

```
public static boolean egituraBerdina (BinTree pZuhaitz1,
                                      BinTree pZuhaitz2)
```

Diseinu errekurtsiboa:

- **Parametroak:** bi zuhaitz <u>return</u> boolean (true egitura berdina badute).
- **Kasu nabaria(k):**

Bi zuhaitzak hutsak -> true.
Zuhaitz bat hutsa eta beste ez -> false.

- **Kasu orokorra(k):**

Inongo zuhaitza hutsa ->
egituraBerdina(1. az. ezkerra, 2. az. ezkerra) AND
egituraBerdina (1. az. eskuina, 2. az. eskuina)

Inplementazioa:

```
public static boolean egituraBerdina (BinTree pZuhaitz1,
                                      BinTree pZuhaitz2) {
    return egituraBerdina1(pZuhaitz1.root pZuhaitz2.root);
}

public static boolean egituraBerdina1(BTNode pB1, BTNode pB2){
   if ( (pB1==null) && (pB2==null) ) return true;
    else
      if ( (pB1==null) || (pB2==null) ) return false;
       else
         return((egituraBerdina1(pB1.left,pB2.left)) &&
                (egituraBerdina1(pB1.right,pB2.right)));
}
```

(I94)
public int adabegiKop(int pMaila)

Diseinu errekurtsiboa:

- **Parametroak:** in zuhaitza**,** in osoa (pMaila) out osoa (zuhaitzako adabegiak maila horretan)
- **Kasu nabaria(k):**

Zuhaitz hutsa -> 0
Zuhaitz ez hutsa eta maila bat bada -> 1 (erroa)

- **Kasu orokorra(k):**
  Zuhaitz ez hutsa eta maila > 1 ->
  adabegiKop1(az ezkerra, maila - 1) + **adabegiKop1**(az. eskuina, maila - 1)

Inplementazioa:

```
public int adabegiKop (int pMaila) {
      return adabegiKop1(root, pMaila);
}
public int adabegiKop1 (BTNode pNode, int pMaila) {
   if (pNode==null) return 0;
    else if ((pNode!=null) && (pMaila==1) ) return 1;
     else return ( (adabegiKop1(pNode.left, pMaila-1))+
                   (adabegiKop1(pNode.right, pMaila-1)) );
}
```

(E95)
`public boolean isHaztatua()`
-- Pre: hutsa
-- Pos: true ssb zuhaitza haztatua badago

Azpizuhaitz bakoitzaren **pisua** jakin behar dugu. `isHaztatua1` metodo laguntzailea definitzen dugu parametro gehiagorekin (murgilketa):

Diseinu errekurtsiboa:

- **Parametroak:** in zuhaitza, out pisua out boolean zuhaitza haztatua den adierazteko. Irteerako bi parametro daudenez, *Context* klase laguntzaile bat sortzen dugu.
- **Kasu nabaria(k):**
  Zuhaitz hutsa -> pisua 0 eta true
- **Kasu orokorra(k):** (zuhaitz ez hutsa)
  CtxEzkerra=isHaztatua1(az. ezkerra)
  CtxEskubia= isHaztatua1(az eskubia)
  Itzuli pisu bezala bi zuhaitzen batura (ctxEskubia.pisua+ctzEzkerra.pisua) gehi erroarena. Booleano bezala true ssb bi azpizuhaitzak haztatuak badira eta beren pisuak berdinak badira.

Inplementazioa:

```
public boolean isHaztatua() {
   Context ctx= isHaztatua1(root);
   return ctx.haztatua;
}

public Context isHaztatua1(BTNode pNode) {
if (pNode==null)  return (new Context(0,true));
 else {
  Context ctxLeft= isHaztatua1(pNode.left);
  Context ctxRight= isHaztatua1(pNode.right);
  Integer v=pNode.content;
  int pisua=ctxLeft.pisua+ctxRight.pisua+v.intValue();
  booleanHaztatua=((ctxLeft.haztatua)&&
                (ctxRight.haztatua )
                && (ctxLeft.pisua==ctxRight.pisua ) );
  return (new Context(pisua,booleanHaztatua));
  }

}
public class Context{
   int pisua;
boolean haztatua;
 public Context(int pPisua, boolean pHaztatua) {
   haztatua=pHaztatua;
   pisua=pPisua;}}
```

(S95)

`public boolean adarOsoa(LinkedList pZerreda)`

Diseinu errekurtsiboa:

- **Parametroak:** in zuhaitza, in zerrenda(`pZerreda`), out boolean
- **Kasu nabaria(k):**

Zuhaitz hutsa → true zerrenda hutsa bada eta false beste kasuan.

Zuhaitz ez hutsa eta zerrenda hutsa → false

Zuhaitz ez hutsa, zerrenda ez hutsa eta erroa zerrend. lehengo elem. desberdin → false

Zuhaitz ez hutsa, elementu bateko zerrenda eta erroaren berdina→ true ssb zuhaitz hostoa

- **Kasu orokorra(k):**

  Zuhaitz ez hutsa, elementu bat baino gehiagoko zerrenda eta erroa zerrendaren lehen elementuaren berdina →

  adarOsoa1(az. ezkerra, zerrenda (-top elementua)) OR **adarOsoa1**(az. eskuina, zerrenda(-top elementua))

Inplementazioa:

```
public boolean adarOsoa(LinkedList pZerreda) {
 return adarOsoa1(pZerreda,bTree.root);
}

public boolean adarOsoa1(LinkedList pZerr, BTNode pNode) {
   if (pNode==null) return !pZerr.hasMoreElements();
     else if (!pZerr.hasMoreElements()) return false;
     else
       {
        String i1=pNode.content;
        String i2=pZerr.getActual();
        if (!i1.equals(i2)) return false;
         else
          {pZerr.goNext();
           if (!pZerr.hasMoreElements ())
             return  ( (pNode.left==null) &&
                       (pNode.right==null));
            else
             return
             ( (adarOsoa1(pZerr,pNode.left)) ||
               (adarOsoa1(pZerr,pNode.right)));
          }
       }
}
```

(E96)

Comenta
Comenta
Eliminad

```
public void etiketatu()
```
--Pre: hutsa
--Pos: “pTree” zuhaitzaren adabegiak, bere sakontasunarekin eta bere seme mota zenbakiarekin izendatura daude (-1=ezkerra, 1=eskuina edo 0=erroa)

Azpizuhaitz bakoitzaren **sakonera** eta adabegi bakoitzaren **seme mota** jakin behar da. Horretatako **etiketatu1** metodo errekurtsiboa definitzen dugu parametro gehiagorekin. (murgilketa):

Diseinu errekurtsiboa:

- **Parametroak**: zuhaitza
- **Kasu nabaria(k):**

Zuhaitz hutsa -> zuhaitz hutsa eta sakontasuna 0

- **Kasu orokorra(k):** (zuhaitz ez hutsa)

  seme mota esleitu erroari
  ctxEzkerra etiketatu1(az. ezkerra, -1)
  ctxEskubia etikettu1(az. eskuina, 1)
  erroari esleitu sakontasun maximoa +1
  erroari esleitutako sakonera itzuli

Inplementazioa

```
public static void etiketatu() {
  Context ctx=etiketatu(root,0);
  }

public static int etiketatu1(BTNode pNode, int pSemeMota)
  {
   if (pNode==null) return new Context(null, 0);
    else {
     (pNode.content).semeMota=pSemeMota;
     int sakEzk=etiketatu1(pNode.left,-1);
     int sakEsk=etiketatu1(pNode.right,1);
     if (sakEzk>sakEsk)
      (pNode.content).sakonera=sakEzk+1;
       else
        (pNode.content).sakonera=sakEsk+1;
     return  ( (pNode.content).sakonera));
    }
  }
```

Con form

(I96)
`public LinkedList adarHandiena()`
--Pos: “pNode” zuhaitzaren adar luzeagoren adabegi guztien zerrenda `LinkedList` bat itzultzen du

Azpizuhaitz bakoitzaren sakontasuna jakin behar da. adarHandiena1 metodo errekurtsibo laguntzaile bat definitzen dugu parametro gehiagorekin. (murgilketa):

Diseinu errekurtsiboa:

- **Parametroak:** out zerrenda (adar luzeagorekin), out osoa (sakontasuna). Irteerako bi parametroak Context klasean biltzen dira
- **Kasu nabaria(k):**
  Zuhaitz hutsa -> zerrenda hutsa eta sakontasuna 0
- **Kasu orokorra(k):** (zuhaitz ez hutsa)
  ctxEzkerra=adarHandiena1(az. ezkerra)
  ctxEskubia=adarHandiena1(az. eskubia)
  Itzuli:
  - Zerrenda handiena, erroaren balioarekin elementu berri bat txertatuz zerrendaren hasieran
  - sakontasun maximoa + 1

Inplementazioa:

```
public class Context {
       int sak;
       LinkedList zerrenda;
}

public LinkedList adarHandiena() {
   Context ctx=adarHandiena1(root);
   return ctx.zerrenda;
  }
```

```
public Context adarHandiena1(BTNode pNode) {
 if (pNode==null)
  return new Context(new LinkedList(),0);
   else {
    Context ctxLeft= adarHandiena1(pNode.left);
    Context ctxRight= adarHandiena1(pNode.right);
    if (ctxLeft.sak>ctxRight.sak)
     {
      ctxLeft.zerrenda.insertFisrt(pNode.content);
      return new Context(ctxLeft.zerrenda,ctxLeft.sak+1);
     }
      else {
       ctxRight.zerrenda.insertFisrt(pNode.content);
       return new
                  Context(ctxRight.zerrenda,ctxRight.prof+1);
      }
    }}
```

(E97)

```
public boolean etiketatuAvl()
```

--Pre: hutsa
--Pos: “pNode” zuhaitzaren adabegiek, orekazko informazioa daukate.
-- metodoak *true* itzultzen du ssb zuhaitza AVL bada.

Azpizuhaitz bakoitzaren **sakonera** jakin behar da. ***etikAvl*** metodo laguntzailea definitzen dugu irteerako parametro gehiagorekin (*Context* klase berri bat definituz)(murgilketa):

Diseinu errekurtsiboa:

- **Parametroak**: in zuhaitza, out sakontasuna (int); out isAVL (boolean);
- **Kasu nabaria(k)**: zuhaitz hutsa -> true, 0
  Kasu orokorra(k): (zuhaitz ez hutsa)
  ctxEzkerra= **etikAvl**(az. ezkerra);
  ctxEskubia= **etikAvl**(az. eskuina);
  erroaren “oreka” aldagaiari esleitu ctxEzkerra.sak – ctxEskuina.sak
  itzuli: max(ctxEzkerra.sak, ctxEskuina.sak)+1,
  ctxEzkerra.isAvl and ctxEskuina.isAvl and
  abs(ctxEzkerra.sak– ctxEskuina.sak)<=1

Inplementazioa:

```
public Context etikAvl(BTNode pNode) {
  if (pNode==null) return new Context(true,0);
   else {
    Context ctxLeft=etiqAvl(pNode.left);
    Context ctxRight=etiqAvl(pNode.right);
    pNode.oreka=ctxRight.sak-ctxLeft.sak;
    int newSak;
    if (ctxLeft.sak>ctxRight.sak)
     newSak=ctxLeft.sak+1;
     else
     newSak=ctxRight.sak+1; //Balio absolutua lortu
    int absSak=ctxLeft.sak-ctxRight.sak;
    if ((ctxLeft.sak-ctxRight.sak) < 0 )
      absSak=(-1)* (ctxLeft.sak-ctxRight.sak);

    boolean isAvl=( (ctxLeft.isAvl) &&
                    (ctxRight.isAvl) &&
                    (absSak <=1) );
    return new Context(isAvl,newSak);

  }}
```

```
public static boolean etiketatuAvl() {
   Context ctx=etikAvl(root);
   return ctx.isAvl;
  }
```

(I97)
`public int hostoEzkMailan(int pMaila)`
-- Pre: hutsa
-- Pos: “pMaila” mailan dauden hostoak eta seme ezkerrak direnak
Azpizuhaitz bakoitzean, erroa **seme ezkerra** den jakin behar dugu. **`hostoEzkMailan1`** metodo laguntzaile bat definitzen dugu parametro gehiagorekin (murgilketa).

Diseinu errekurtsiboa:

- **Parametroak**: in adabegia, in maila, in boolean (es izdo), return int (baldintzak betetzen dituzten hosto kopurua)
- **Kasu nabaria(k)**:

Zuhaitz hutsa -> 0
maila= 1 eta hosto zuhaitza eta ezkerra da -> 1
maila=1 eta (zuhiatza ez hostoa edo ez da ezkerra) ->0

- **Kasu orokorra(k)**: (zuhaitz ez hutsa eta maila 1 baino handiagoa)
  itzuli **`hostoEzkMailan1`** (az. ezkerra, maila-1,true) +
  `hostoEzkMailan1` (az. eskuina, maila-1,false)

Inplementazioa:

```
public int hostoEzkMailan(int pMaila)
{ return hostoEzkMailan1(root,pMaila,false);}

public int hostoEzkMailan1(BTNode pNode, int pMaila,boolean
pEzk) {
if (pNode==null) return 0;
 else
  if (pMaila==1)
   if ((pNode.left==null) && (pNode.right==null) && pEzk)
    return 1;
     else
     return 0;
  else
  return hostoEzkMailan1(pNode.left,pMaila-1,true)+
         hostoEzkMailan1(pNode.right,pMaila-1,false);
  }
```

(E98)
`public boolean biOrekatua()`
--Pre: hutsa
--Pos: true ssb zuhaitza 2-orekatua bada
Azpizuhaitz bakoitzaren 2 graduko adabegi kopurua jakin behar dugu. **is2Oreka** metodo laguntzaile errekurtsiboa definitzen dugu parametro gehiagorekin (murgilketa):

Diseinu errekurtsiboa:
Parametroak: in zuhaitza, out boolean (bi orekatua den ala ez), out osoa (2 graduko adabegi kopurua). Irteerako bi parametro biltzeko *Context* klase berri bat definitzen dugu (parametroak is2ork eta zGradu2 dira).

Kasu nabaria(k):
Zuhaitz hutsa -> true eta 0

- **Kasu orokorra(k)**: (zuhaitz ez hutsa)
  ctxEzkerra= **is2Oreka** (az. ezkerra)
  ctxEskuina= **is2Oreka**(az. ezkuina)
  itzuli:
  * true ssb ctxEzkerra. is2ork eta ctxEskuina. is2ork eta ctxEzkerra.zGradu2 = ctxEskuina.zGradu2
  * ctxEzkerra.zGradu2+ctxEskuina.zGradu2 azpizuhaitzaren bat hutsa bada eta ctxEzkerra.zGradu2+ ctxEskuina.zGradu2+ 1 kontrako kasuan

Inplementazioa:

```
public boolean biOrekatua() {
 Context ctx=is2Oreka(root);
 return ctx.is2ork;
}

public Context is2Oreka(BTNode pNode) {
if (pNode==null) return new Context(true,0);
 else {
  Context ctxLeft= is2Oreka(pNode.left);
  Context ctxRight= is2Oreka(pNode.right);
  boolean is2ork=ctxLeft.is2ork && ctxRight.is2ork &&
                 ctxLeft.zGradu2==ctxRight.zGradu2;
  int zgradu2;
  if ((pNode.left==null) || (pNode.right==null))
    zgradu2=ctxLeft.zGradu2+ctxRight.zGradu2;
   else
    zgradu2=ctxLeft.zGradu2+ctxRight.zGradu2+1;
  return new Context(is2ork,zgradu2);
}
}
```

(I98)

```
public static boolean isAurrizkia(BinTree pT1, BinTree pT2)
```

Diseinu errekurtsiboa:

- **Parametroak:** in zuhaitza(pT1), in zuhaitza(pT2), return boolean
- **Kasu nabaria(k):**

zuhaitz 1 hutsa-> true
zuhaitz 1 ez hutsa eta zuhaitz 2 hutsa -> false
bi zuhaitzak ez hutsak eta erro desberdina -> false

- **Kasu orokorra(k):** (bi zuhaitzak ez hutsak eta erro berdina)
  itzuli **aurrizki**(az1 ezkerra, az2 ezkerra) and **aurrizki**(az1 eskuina, az2 eskuina)

Inplementazioa:

```
public static boolean isAurrizkia(BinTree pT1, BinTree pT2)
{
 return (isAurrizkia11(pT1.root, pT2.root));
}

public static boolean isAurrizkia1(BTNode pNode1, BTNode
pNode2) {
  if (pNode1==null) return true;
   else if (pNode2==null) return false;
    else
     {
      Integer i1=pNode1.content;
      Integer i2=pNode2.content;

      if (i1.intValue()!=i2.intValue()) return false;
     else
      return ( (isAurrizkia1(pNode1.left,pNode2.left)) &&
               (isAurrizkia1(pNode1.right,pNode2.right)) );

  }
  }
```

(1)

`public int azpizuhaitzHutsBakarra(int pNumAurre)`

Diseinu errekurtsiboa:

- **Parametroak**: in zuhaitza, in osoa (emandako parametroa), return int
- **Kasu nabaria(k)**
  Zuhaitz hutsa -> 0
  Zuhaitz hostoa eta `pNumAurre`=0 -> 1
  Zuhaitz hutsa eta `pNumAurre`>0 -> 0
  Zuhaitz ez hutsa ezta hostoa, azpizuhaitz huts batekin eta `pNumAurre`=0 -> 0
- **Kasu orokorra(k)**: (zuhaitz ez hutsa ez hostoa eta (azpizuhaitz huts bat badauka eta `pNumAurre` >0))
  Zuhaitz ez hutsa, az. ezkerra ez hutsa eta az. eskuina hutsa ->
  `azpizuhaitzHutsBakarra1` (azpEzk, `pNumAurre`-1)
  Zuhaitz ez hutsa, az. ezkerra hutsa eta az. eskuina ez hutsa ->
  `azpizuhaitzHutsBakarra1` (azpEsk, `pNumAurre`-1)
  zuhaitz ez hutsa eta 2 azpizuhaitz ez hutsak ->
  `azpizuhaitzHutsBakarra1` (azpEzk, `pNumAurre`) +
  `azpizuhaitzHutsBakarra1` (azpEsk, `pNumAurre`)

Inplementazioa

```
public int azpizuhaitzHutsBakarra(int pNumAurre) {
   return azpizuhaitzHutsBakarra1(root,pNumAurre);
}

public int azpizuhaitzHutsBakarra1(BTNode pNode,
                                              int pNumAurre) {
  if (pNode==null) return 0;
   else
   if((pNode.left==null)&& (pNode.right==null) )
    if (pNumAurre==0)return 1;else return 0;
     else if (((pNode.left==null) ||
                (pNode.right==null)) && pNumAurre==0)
      return 0;
    else if (pNode.right==null)
    return azpizuhaitzHutsBakarra1(pNode.left,pNumAurre-1);
      else if (pNode.left==null)
       return
         azpizuhaitzHutsBakarra1(pNode.right,pNumAurre-1);
       else
        return
         azpizuhaitzHutsBakarra1(pNode.left,pNumAurre)+
         azpizuhaitzHutsBakarra1(pNode.right,pNumAurre);
}
```

(2)

```
public LinkedList hostoZerrenda(int pNumAurre,
                                 LinkedList pZerrenda)
```

Diseinu errekurtsiboa:

- **Parametroak**: in zuhaitza, in osoa , out zerrenda
- **Kasu nabaria(k)**:
  Zuhaitz hutsa -> zerrenda hutsa
  Zuhaitz hostoa eta `pNumAurre`= 0 -> zerrenda adabegi hostoarekin
  Zuhaitz hostoa eta `pNumAurre`>0-> zerrenda hutsa
  Zuhaitz ez hutsa ezta hostoa azpizuhaitz huts batekin eta `pNumAurre`=0 -> zerrenda hutsa
- **Kasu orokorra(k)**: (Zuhaitz ez hutsa ezta hostoa eta (azpizuhaitz huts bat badauka eta `pNumAurre` >0))
  Zuhaitz ez hutsa, az. ezkerra ez hutsa eta az. eskuina hutsa ->
  itzuli **hostoZerrenda1**(azpEzk, `pNumAurre`-1, zerrenda)
  Zuhaitz ez hutsa, az. ezkerra hutsa eta az. eskuina ez hutsa ->
  itzuli **hostoZerrenda1**(azpEsk, `pNumAurre`-1, zerrenda)
  zuhaitz ez hutsa eta 2 azpizuhaitz ez hutsak ->
  zerrenda1= **hostoZerrenda1**(azpEzk, `pNumAurre`, zHutsa);
  zerrenda2= **hostoZerrenda1**(az eskuina, `pNumAurre`, zHutsa);
  itzuli zerrenda1+zerrenda2

Inplementazioa

```
public LinkedList hostoZerrenda(int pNumAurre,
                                 LinkedList pZerrenda) {

 return hostoZerrenda1(root,pNumAurre, pZerrenda);

}

public LinkedList hostoZerrenda1(BTNode pNode,
                int pNumAurre, LinkedList pZerrenda) {
if (pNode==null) return new LinkedListList();
 else if((pNode.left==null)&& (pNode.right==null) )
  if (pNumAurre==0){
   pZerrenda.insert(pNode.content);
   return pZerrenda;
  } else return new LinkedList();
  else if ( ((pNode.left==null)||
            (pNode.right==null)) && (pNumAurre==0) )
     return new LinkedList();
     else if (pNode.right==null)
      return hostoZerrenda1(pNode,pNumAurre-1, pZerrenda);
       else if (pNode.left==null)
        return hostoZerrenda1(pNode.right,pNumAurre-1,
                                          pZerrenda);
         else {
          LinkedList lLeft= hostoZerrenda1(pNode.left,
                                pNumAurre, new LinkedList());
          LinkedList lRight= hostoZerrenda1(pNode.right,
                                pNumAurre, new LinkedList());
       // link metodoak bi zerrendak batzen ditu
       LinkedList berria= LinkedList.link(lLeft,lRight);
       return berria;
     }
}
```

(3)

`public int zOsoakMailan(int pMaila, int pBalioa)`

Diseinu errekurtsiboa:

- **Parametroak:** in zuhaitza, in osoa (pMaila), in osoa(pBalioa) return osoa
- **Kasu nabaria(k):**

Zuhaitz hutsa -> 0
Zuhaitz ez hutsa, `pMaila`=0 eta erroa ez dauka pBalioa -> 0
Zuhaitz ez hutsa, `pMaila`=0 eta erroa pBalioa balioa dauka -> 1

- **Kasu orokorra(k):** (zuhaitz ez hutsa eta maila>0)

itzuli **zOsoakMailan1**(azpEzk, pMaila-1, pBalioa) +
zOsoakMailan1(azpEsk, pMaila-1, pBalioa)

Inplementazioa:

```
public int zOsoakMailan(int pMaila,int pBalioa) {
   return zOsoakMailan1(root, pMaila,pBalioa);
}
```

```
public static int zOsoakMailan1(BTNode pNode,int pMaila,
                                int pBalioa) {
  if (pNode==null) return 0;
   else
   if (pMaila==0)
     if (pNode.content).intValue()!=pBalioa)
      return 0;
     else return 1;
   else

  return zOsoakMailan1(pNode.left,pMaila-1,pBalioa)+
        zOsoakMailan1(pNode.right,pMaila-1,pBalioa);
  }
```

(5)

`public BinTree ispiluIrudia()`

Diseinu errekurtsiboa:

- **Parametroak:** in zuhaitza1 out zuhaitza2
- **Kasu nabaria(k):**

Zuhaitz hutsa -> zuhaitza

- **Kasu orokorra(k):** (zuhaitz ez hutsa)
  Az. eskubia esleitu **`ispiluIrudia1`** (azpEzk)
  Az. ezkerra esleitu **`ispiluIrudia1`** (azpEsk)

Inplementazioa

```
public BinTree ispiluIrudia() {
 BinTree b=new BinTree();
 b.root= ispiluIrudia1(this.root);
 return b;
}

public BTNode ispiluIrudia1(BTNode pNode) {
    if (pNode!=null) {
     BTNode node=new BTNode(pNode.content);
     node.left=pNode.right);
     node.right=pNode.left);
     return node;
     } else return pNode;
}
```

(6)
public static boolean isIspilua(BinTree pNode1, BinTree pNode2)
Diseinu errekurtsiboa:

- **Parametroak:** in 2 zuhaitz return boolean
- **Kasu nabaria(k):**

2 zuhaitz hutsak -> *true*
zuhaitz bat hutsa eta bestea ez -> *false*
2 zuhaitz ez hutsak eta erro desberdina -> *false*

- **Kasu orokorra(k):** (2 zuhaitz ez hutsak erro berdinekin)
  itzuli **isIspilua1**(azpEzk, azpEsk) and
  isIspilua1(azpEsk, azpEzk)

Inplementazioa

```
public static boolean isIspilua(BinTree pTree1, BinTree pTree2) {
  return isIspilua1(pTree1.root,pTree2.root);
}

public static boolean isIspilua1(BTNode pNode1, BTNode pNode2) {
 if ( (pNode1==null) && (pNode2==null) ) return true;
  else if ( (pNode1==null) || (pNode2==null ) ) return false;
   else {
    Integer i1=pNode1.content;
    Integer i2=pNode2.content;
    if (i1.intValue()!=i2.intValue() ) return false;
    else
     return isIspilua1(pNode1.left,pNode2.left) &&
            isIspilua1(pNode1.right,pNode2.right);
}
}
```

(8)
`public int luzeeraMax(int pBalioa)`
--Pos: “pBalioa” agertzen den aldi kopuru maximoaren adarraren luzeera itzuliko du

Azpizuhaitz bakoitzean **elementuaren errepikapen kopuru maximoa adar berberan** behar dugu. **`luzMax`** metodo errekurtsibo laguntzailea definitzen dugu parametro gehiagorekin (murgilketa). Irteerako parametroak *Context* klasean biltzen ditugu.

Diseinu errekurtsiboa:

- **Parametroak**: in zuhaitza; in osoa (n), out osoa (luz); out osoa (n errepikapenak, gehien agertzen den adarrean (errep))
- **Kasu nabaria(k)**:

Zuhaitz hutsa -> 0,0

- **Kasu orokorra(k):** (zuhaitz ez hutsa)

(luz ezkerra, errep ezkerra) **=luzMax**(azpEzk, n);
(luz eskubia, errep eskubia**) =luMax**(azpEsk, n);
errep = max(errep ezkerra, errep eskuina) + (1 erroa “n” badauka)
luz azpizuhaitzaren errepikapen handienarekin +1

Inplementazioa

```
public int luzeeraMax(int pBalioa) {
   Context ctx=luzMax(root, pBalioa);
   return ctx.luz;
}
public Context luzMax(BTNode pNode, int pBalioa) {
if (pNode==null) return new Context(0,0);
 else
 {
  Context ctxLeft=luzMax(pNode.left,pBalioa);
  Context ctxRight=luzMax(pNode.right,pBalioa);
  int newLuz;
  int newErrep;
  if (ctxLeft.errep>ctxRight.errep)
   {
    newLuz=ctxLeft.luz+1;
    newErrep=ctxLeft.errep;
   }
     else
     {
      newLuz=ctxRight.luz+1;
      newErrep=ctxRight.errep;
     }
if ( (pNode.content).intValue()==pBalioa)
     newErrep++;
return new Context(newErrep,newLuz)}}
```

(12)

`public boolean isBzb()`

--Pre: hutsa

--Pos: true ssb bilaketazko zuhaitz bitar bat bada

Sarrera bezala, **eremu maximo eta minimo** bat eman behar zaie azpizuhaitzean dauden elementu osoei. **`isBzb1`** metodo errukurtsibo laguntzailea definitzen dugu parametro gehiagorekin. (murgilketa):

Diseinu errekurtsiboa:

- **Parametroak**: in zuhaitza; in osoa (eremuMin); in osoa (eremuMax.); return boolean
- **Kasu nabaria(k)**:

Zuhaitz hutsa -> true

Zuhaitz ez hutsa eta ez da gertatzen (eremuMin <erroa eta erroa <eremuMax) -> false

- **Kasu orokorra(k):** (zuhaitz ez hutsa eta erroa sarrerako bi eremuen artean dago)
  itzuli **`isBzb1`**(azpEzk,eremuMin,erroa) and
  `isBzb1` (azpEsk, erroa, eremuMax)

Inplementazioa:

```
public boolean isBzb() {
      return isBzb1(root,1000, -1000);
  }

public  boolean  isBzb1(BTNode  pNode,  int  eremuMax,  int
eremuMin) {
    if (pNode==null) return true;
     else {
     int i=(pNode.content).intValue();
     if ( !(eremuMin<i && i<eremuMax) )
      return false;
       else
        return isBzb1(pNode.left,i,eremuMin) &&
               isBzb1(pNode.right,eremuMax,i);

  }}
```

(13)

```
public int sakonera()
```

Diseinu errekurtsiboa:

- **Parametroak:** in zuhaitza, return osoa
- **Kasu nabaria(k):**

Zuhaitz hutsa -> 0

- **Kasu orokorra(k):** (zuahitz ez hutsa)

```
Sakon1 sakonera(azpEzk)
sakon2= sakonera(azpEsk)
```

Itzuli bietatik sakontasun handiena+ 1 (erroarengatik)

Inplementazioa:

```
public int sakonera() {
   return sakonera1(root);
}
```

```
 public int sakonera1(BTNode pNode) {
   if (pNode==null) return 0;
    else {
     int p1= sakonera1(pNode.left);
     int p2= sakonera1(pNode.right);
     if (p1>p2) return p1+1;
      else return p2+1;
    }
}
```