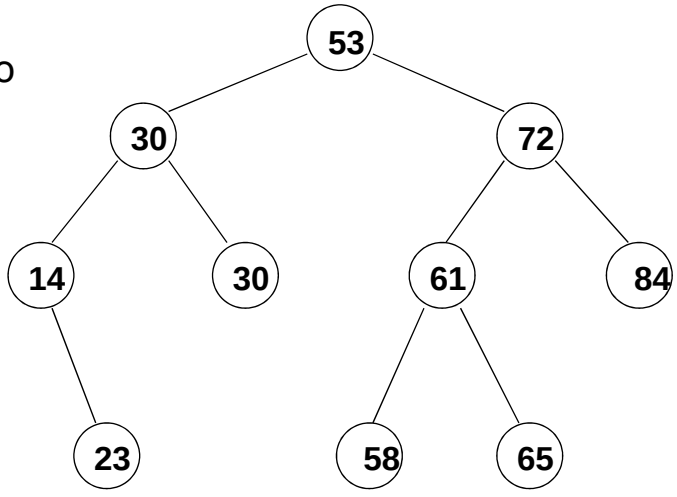
Bilaketa-zuhaitz bitarrak

Bilaketa-zuhaitz bitarra

- Zuhaitz bitarra da. Adabegi bakoitzean, ezkerreko azpizuhaitzeko elementu guztiak erroko balioa baino txikiagoak dira, eta eskuineko azpizuhaitzeko adabegi guztiak erroko balioa baino handiagoak edo berdinak izango dira



Bilaketa-zuhaitz bitarren eragiketak

- Zuhaitz bitarren eragiketa berdinak, gehi beste hauek:

Eragiketa	Deskribapena
void addElement (T elem);	Elementua zuhaitzean gehitzen du
T removeElement(T elem)	Emandako elementua zuhaitzetik kentzen du, eta elementu hori bueltatzen du
void removeAllOcurrences(T elem)	Elementu horren agerpen guztiak ezabatuko ditu
T removeMin()	Zuhaitzeko elementu minimoa ezabatuko du
T removeMax()	Zuhaitzeko elementu handiena ezabatuko du
T findMin()	Zuhaitzeko elementu minimoa bueltatuko du
T findMax()	Zuhaitzeko elementu maximoa bueltatuko du

Bilaketa-zuhaitz bitarraren interfazea

```
public interface TADBinarySearchTree<T> extends TADBinaryTree<T>{

    // "elem" zuhaitzean gehituko du
    public void addElement(T elem);

    // "elem" zuhaitzetik kenduko du
    // eta objektuaren erreferentzia bueltatuko du
    // (null elementua ez bazegoen)
    public T removeElement (T elem);

    // elementuaren .equals betetzen duten adabegi guztiak kenduko ditu
    public void removeAllOcurrences (T elem);

    // zuhaitzaren elementu minimoa ezabatuko du, eta beraren erreferentzia bueltatuko du
    // Aurrebaldintza: zuhaitza ez da hutsa
    public T removeMin ();

    // zuhaitzaren elementu maximoa ezabatuko du, eta beraren erreferentzia bueltatuko du
    // Aurrebaldintza: zuhaitza ez da hutsa
    public T removeMax();

    // zuhaitzaren elementu minimoaren erreferentzia bueltatuko du
    // Aurrebaldintza: zuhaitza ez da hutsa
    public T findMin();

    // zuhaitzaren elementu maximoaren erreferentzia bueltatuko du
    // Aurrebaldintza: zuhaitza ez da hutsa
    public T findMax();
}
```

Comparable interfazea inplementatu behar du

Bilaketa-zuhaitz bitarrentzako klasea

```
public class BinaryTreeNode<T>
{
  protected T element;
  protected BinaryTreeNode<T> left, right;
  ...
}
```

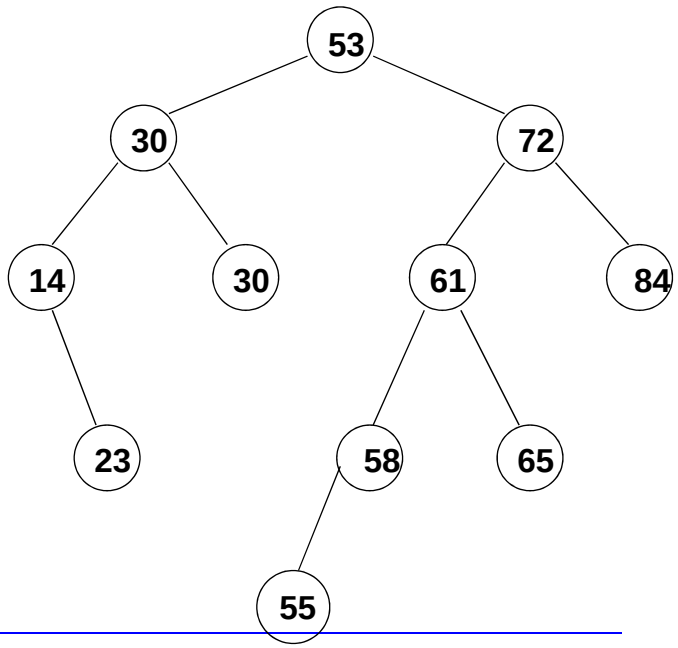

Bilaketa-zuhaitz bitarrentzako klasea

Comparable interfazea inplementatu behar du

```
public class LinkedBinSearchTree<T>  extends LinkedBinaryTree<T>
                                     implements TADBinarySearchTree<T>{
   // protected BinaryTreeNode<T> root;
   // protected int count;
   /** Creates an empty binary search tree. */
  public LinkedBinarySearchTree()
  {  super();
  }
  /** Creates a binary search with the specified element as its root.
   * @param element  the element that will be the root of the new
    binary search tree */
  public LinkedBinarySearchTree (T element)
  {  super (element);
  }
```

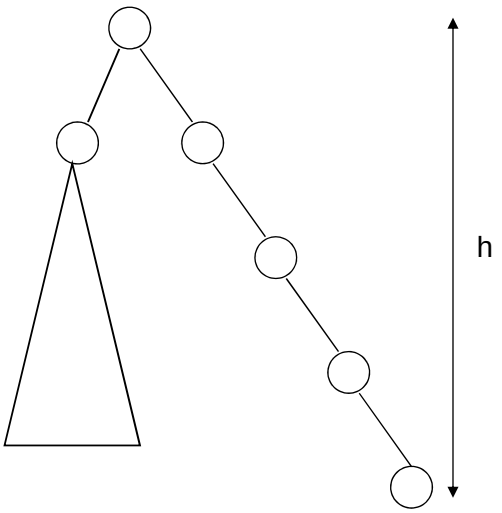
void addElement(T elem)

```
public void addElement (T element)  {
    BinaryTreeNode<T> temp = new BinaryTreeNode<T> (element);
    Comparable<T> comparableElement = (Comparable<T>)element;

    if (isEmpty())  root = temp;
    else
    { BinaryTreeNode<T> current = root;
      boolean added = false;

      while (!added)  {
        if (comparableElement.compareTo(
                          current.element) < 0)
        {   if (current.left == null)
             {  current.left = temp;
                added = true;
             }
            else current = current.left;
        }
        else { if (current.right == null)
                    {  current.right = temp;
                       added = true;
                    }
               else current = current.right;
        }
      } // while
    } // if
    count++;
}
```



Analisia

- Adar bakarra aztertu behar dugu, elementuaren posizioa aurkitu arte. Zuhaitzaren eraiketan beste murrizpenik ez badago, metodo hau O(n) izango da kasu txarrenean, n zuhaitzeko elementu-kopurua izanda
- Zehaztasun gehiago emanda, esan dezakegu O(h) dela esan dezakegu, h zuhaitzaren altuera izanda
- Zuhaitza orekatuta badago, metodoa O(log n) izango da. Nola lortu zuhaitz orekatua?



T find(T elem). Kasuen azterketa

// elem duen adabegia bueltatuko du.
// null bueltatuko du elem zuhaitzean ez badago
public T find (T elem)

- Oinarrizko kasuak:
  - Zuhaitz hutsa: return null
  - elem erroan dago: return "erroko balioa"
- Kasu orokorra:
  - elem ez dago erroan:
    - Erroko balioa elem baino handiagoa bada, orduan bilatu ezkerreko azpizuhaitzean
    - Erroko balioa elem baino txikiagoa bada, orduan bilatu eskuineko azpizuhaitzean
- Adierazpen honekin, inplementazio errekurtsibo batek azpizuhaitzaren erroa pasa behar du parametroen artean

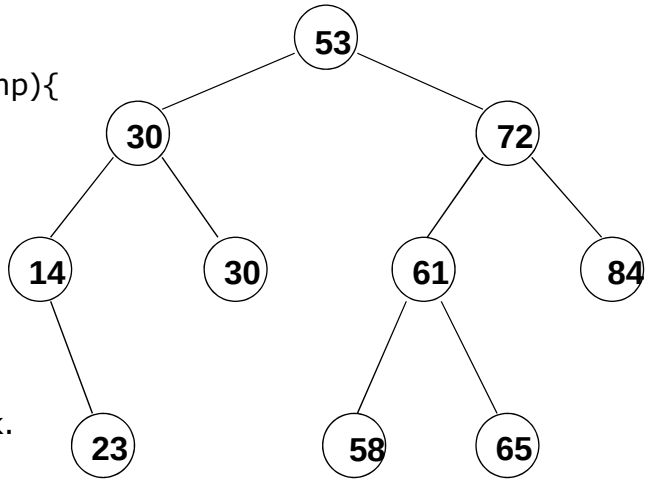
T find(T elem), murgilketarekin

```
public T find(T elem) {
    return find(elem, root);
}

private T find(T elem, BinaryTreeNode<T> adabegi){
    Comparable<T> konp;
    if (adabegi == null)
            return null;
    else {
              konp = (Comparable<T>) adabegi.element;
              if (konp.compareTo(elem) == 0)  return adabegi.content;
              else if (konp.compareTo(elem)>0) return find(elem, adabegi.left);
              else return find(elem, adabegi.right);
    }
}
```

- Analisia: kasurik txarrenean, elementua adabegia-ren azpizuhaitz bakarrean bilatuko da, erroarekin konparatu ondoren. Horregatik, O(h), h zuhaitzaren altuera izanik

T findMin()

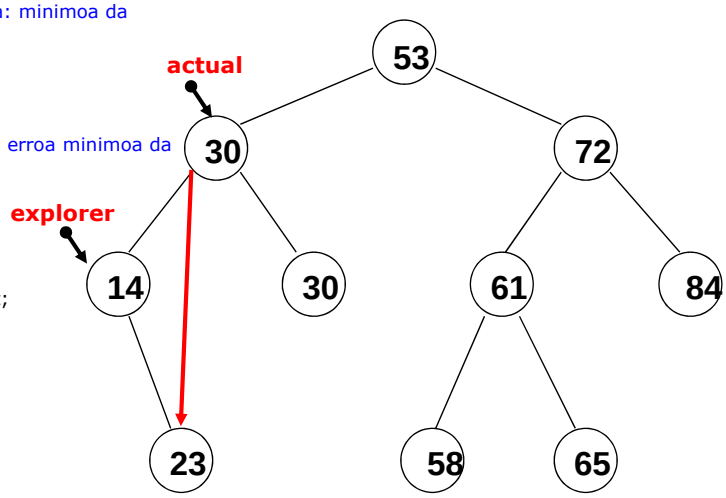
```
// zuhaitza ez da hutsa
public T findMin() {
    return findMin(root);
}

private T findMin(BinaryTreeNode<T> temp){
    if (temp.left == null)
    return temp.element;
    else
        return findMin(temp.left);
}
```



Analisia: O(h) h zuhaitzaren altuera izanik.

T removeMin()

```
// zuhaitza ez da hutsa
public T removeMin() {
      BinaryTreeNode<T> actual = root;
      if (actual.left == null)
            if (actual.right == null){ // bakarrik erroa: minimoa da
                T temp = actual.element;
                root = null;
                count = 0;
                return temp;
            }
            else { // bakarrik eskuineko azpizuhaitza: erroa minimoa da
                T temp = actual.element;
                root = root.right;
                count--;
                return temp;
            }
      else{
            BinaryTreeNode<T> explorer = actual.left;
            while (explorer.left != null){
                actual = actual.left;
                explorer = actual.left;
            }
            T temp = explorer.element;
            actual.left = explorer.right;
            count--;
            return temp;
      }
}
```



Analisia: O(h) h zuhaitzaren altuera izanik.

T removeMin() errekurtsibitatea erabiliz

// balio minimoko elementua ezabatu eta bueltatu.
// aurrebaldintza: zuhaitz ez hutsa.
public T removeMin()

- Oinarrizko kasuak:
  - Zuhaitzak elementu bakarra du (erroa): kendu eta bueltatu balioa
  - Zuhaitzak ezkerreko azpizuhaitz hutsa du eta eskuinekoa ez da hutsa: kendu erroa (eskuineko azpizuhaitza utzita) eta bere balioa bueltatu
- Kasu orokorra:
  - Zuhaitzak ezkerreko azpizuhaitz ez hutsa du: removeMin() ezkerreko azpizuhaitzean

T removeMin(), bertsio errekurtsiboaren lehen saioa

```
// zuhaitz ez hutsa
public T removeMin(){
    return removeMin(root);
}

private T removeMin(BinaryTreeNode<T> adabegi){
    if (adabegi.left == null)
            T temp = adabegi.content;
            adabegi = adabegi.right;
            return temp;
    else
            return removeMin(adabegi.left);
}
```

Hau da bueltatu behar den balioa, baina zuhaitzetik kendu behar da.

Eta adabegi=adabegi.right egiten badugu?

EZ, horrela parametro formalaren balioa aldatzen da (aldagai lokala bezalakoa da), baina ez du aldatzen parametro errealaren balioa

Parametroak aldagai lokalak bezalakoak dira

removeMin(root);

Demagun:

private T removeMin(BinaryTreeNode<T> adabegi){
    .... adabegi = adabegi.right; **....** }

Hau egiten du:

Baina root-etik lortzen den zuhaitza ez da aldatu

T removeMin(), bertsio errekurtsiboa murgilketarekin

```
// zuhaitz ez hutsa
public T removeMin(){
    Emaitza ema= removeMin(root);
    root = ema.adabegia; // ZUHAITZA ALDATZEN DU
    return ema.balioa;
}
```

Zuhaitza aldatzen denean, adabegi baten erreferentzia bueltatu behar da (zuhaitza aldaketaren ondoren)

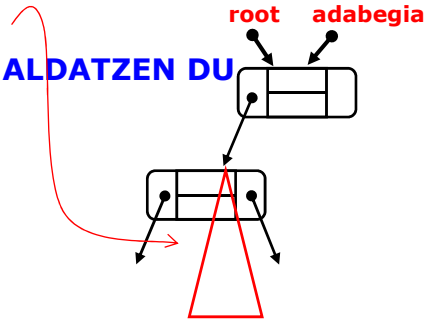
removeMin(adabegia)
"adabegia" erroa duen zuhaitzean minimoa ezabatutakoan geratzen den zuhaitzaren erroaren erreferentzia bueltatuko du.
Gainera, ezabatutako elementuaren balioa bueltatuko du

```
private class Emaitza{
    BinaryTreeNode<T> adabegia;
    T balioa;

    public Emaitza(BinaryTreeNode<T> pAdabegia, T elem){
            adabegia = pAdabegia;
            balioa= elem;
    }
}
```

Emaitza removeMin(adabegia)

removeMin(adabegia)
"adabegia" erroa duen zuhaitzean minimoa ezabatutakoan geratzen den zuhaitzaren erroaren erreferentzia bueltatuko du. Gainera, ezabatutako elementuaren balioa bueltatuko du

```
private Emaitza removeMin(BinaryTreeNode<T> adabegia){
    if (adabegia.left == null)
          return new Emaitza(adabegia.right, adabegia.element);
    else{
          Emaitza ema= removeMin(adabegia.left);
      adabegia.left = ema.adabegia;//ZUHAITZA ALDATZEN DU
      ema.adabegia = adabegia;
      return ema;
    }
}
```



- Analisia: O(h) h zuhaitzaren altuera izanik

T removeElement(T elem). Kasuen azterketa

// Zuhaitzetik emandako elementua ezabatuko du
public T removeElement(T elem)

- Oinarrizko kasuak:
  - Zuhaitz hutsa: null bueltatu
  - elem erroan dago:
    - Erroa hostoa bada, orduan kendu, zuhaitz hutsa utziz, eta balioa bueltatu
    - Eskuineko azpizuhaitza badu, eta ez dago ezkerreko azpizuhaitzik, orduan erroa kendu eta bere balioa bueltatu, eskuineko azpizuhaitza utziz
    - Ezkerreko azpizuhaitza badu, eta ez dago eskuineko azpizuhaitzik, orduan erroa kendu eta bere balioa bueltatu, ezkerreko azpizuhaitza utziz
    - Bi azpizuhaitzak baldin badaude, kendu minimoa eskuineko azpizuhaitzetik eta erroaren tokian ipini
- Kasu orokorra:
  - Erroaren balioa elem baino handiagoa bada: removeElement(elem) ezkerreko azpizuhaitzean
  - Erroaren balioa elem baino txikiagoa bada: removeElement(elem) eskuineko azpizuhaitzean

T removeElement(T elem)

```
public T removeElement(T elem) {
    Emaitza temp = removeElement(elem, root);
    root = temp.adabegia;
    return temp.balioa;
}
```


Emaitza removeElement(elem, adabegia)

```
/** "adabegia" erroa duen azpizuhaitzean "elem" kendu ondoren geratzen den
      zuhaitzaren erroaren erreferentzia bueltatzen du
* Gainera, ezabatutako elementuaren erreferentzia bueltatzen du.  */
private Emaitza removeElement(T elem, BinaryTreeNode<T> adabegia) {
    Comparable<T> konp = (Comparable<T>) elem;
    if (adabegia != null ){
            T adabegiarenBalioa = adabegia.element;
            if (konp.compareTo(adabegiarenBalioa) == 0){  // elem erroan dago
                if (adabegia.left == null && adabegia.right == null ){ // adabegia hostoa da
                    count = 0;
                    return new Emaitza(null, adabegiarenBalioa);
            }
            else if (adabegia.left == null && adabegia.right != null ){ // ezkerrekoa hutsa eta eskuinekoa ez
                    count--;
                    return new Emaitza(adabegia.right, adabegiarenBalioa);
            }
            else if (adabegia.left != null && adabegia.right == null ){// eskuinekoa hutsa eta ezkerrekoa ez
                    count--;
                    return new Emaitza(adabegia.left, adabegiarenBalioa);
            }
            else { // ezkerreko azpizuhaitz ez hutsa eta eskuinekoa ere ez
                    Emaitza ema = removeMin(adabegia.right);
                    adabegia.right = ema.adabegia; // HEMEN ZUHAITZA ALDATZEN DA
                    T erroBerria = ema.balioa;
                    adabegia.element = erroBerria;
                    count--;
                    return new Emaitza(adabegia, erroBerria);
                }
            }//else elem ez dago erroan
```

```
        else    // elem ez dago erroan
            if (konp.compareTo(adabegiarenBalioa) < 0){ // "azpizuhaitzaren erroa > elem"
                Emaitza ema = removeElement(elem, adabegia.left);
                adabegia.left = ema.adabegia; // HEMEN ZUHAITZA ALDATZEN DA
                 ema.adabegia = adabegia;
                  return ema;
            }
            else{
                 Emaitza ema = removeElement(elem, adabegia.right);
                adabegia.right = ema.adabegia; // HEMEN ZUHAITZA ALDATZEN DA
                 ema.adabegia = adabegia;
                return ema;
            }
        }
        else // adabegia== null
                return new Emaitza(null, null);
}
```

- Analisia: O(h) h zuhaitzaren altuera izanik

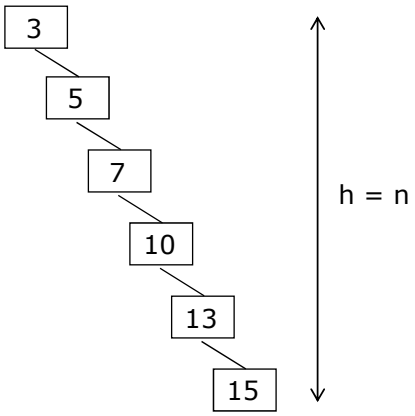
void addElement(T elem) Kasuen analisia

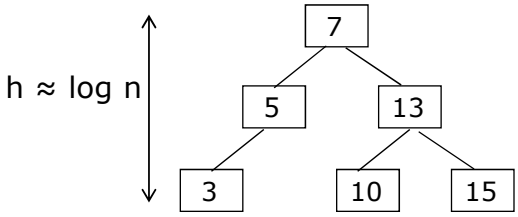
// Elementu bat gehitzen du bilaketa-zuhaitzean
public void addElement (T elem)

- Oinarrizko kasua:
  - Zuhaitz hutsa: elem erroan duen zuhaitza sortu
- Kasu orokorra:
  - Erroaren balioa > elem: gehitu ezkerreko azpizuhaitzean
  - Erroaren balioa <= elem: gehitu eskuineko azpizuhaitzean

Bilaketa-zuhaitz bitar orekatuak

- Elementuen txertaketa sinpleak zuhaitz endekatua sor dezake
- Baina zuhaitz orekatua lor genezake

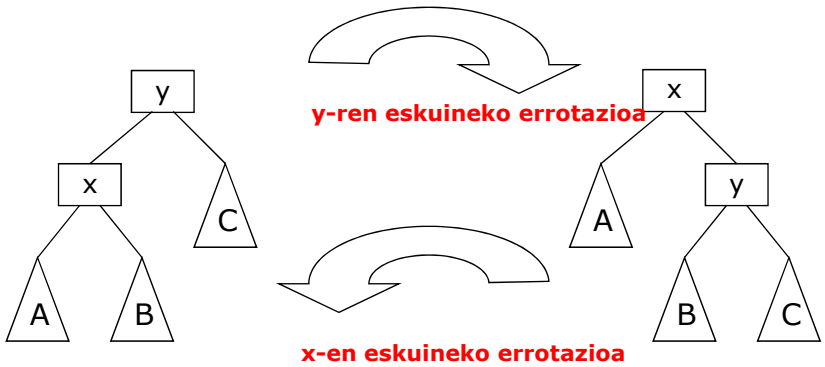




- Bilaketa-zuhaitz bitarra orekatua izango da adabegi guztietan bere azpizuhaitzen altueren arteko diferentzia 1 edo txikiagoa bada

Errotazioaren bidezko oreka

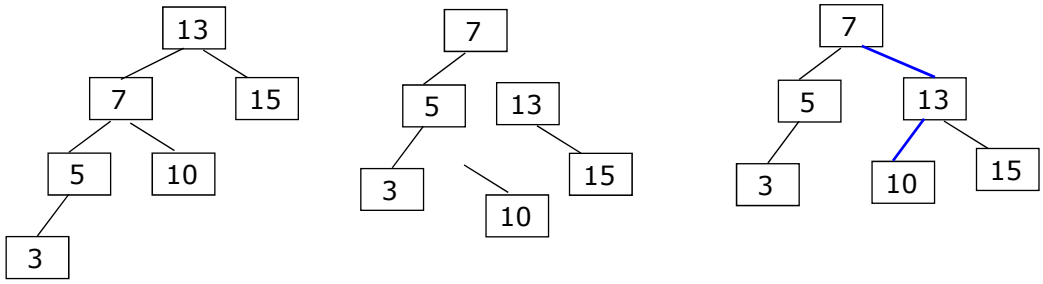
- Bilaketa-zuhaitz bitar orekatu batek oreka galdu dezake adabegi bat txertatu edo ezabatu egiten denean
- Oreka berreskuratu daiteke errotazio egokia egiten bada



(errotazioa zuhaitzaren edozein adabegiri aplikatu ahal zaio)

Eskuineko errotazioaren adibidea

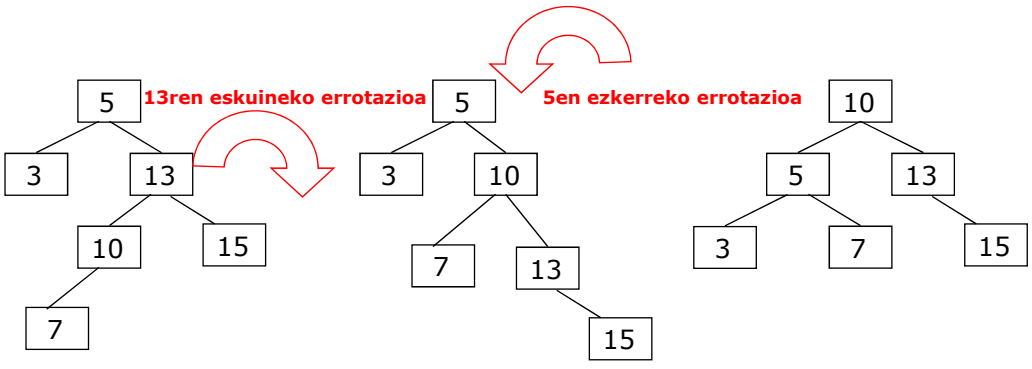
- 13-ren ezkerreko azpizuhaitzak 2ko altuera du, eta eskuinekoak 0
- Desoreka 3 balioko adabegia gehitzean sortu zitekeen
- Erroa eskuinera mugitu (errotazioa)



(Berdin ezkerreko errotazioan)

Eskuin-ezkerreko errotazioa

- Adar luzea eskuineko umearen ezkerreko azpizuhaitzean badago, orduan bi errotazio sinple egin behar dira:
  - Lehenengo eskuineko errotazioa, erroaren eskuineko umeari
  - Ondoren erroaren ezkerreko errotazioa
- Mota honetako errotazioa egingo da 7 balioa duen adabegiaren txertaketaren ondoren



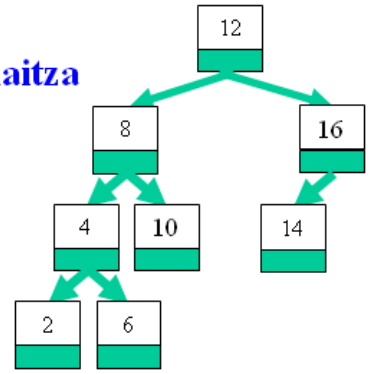
(Berdin ezker-eskuineko errotazioan)

AVL zuhaitzak

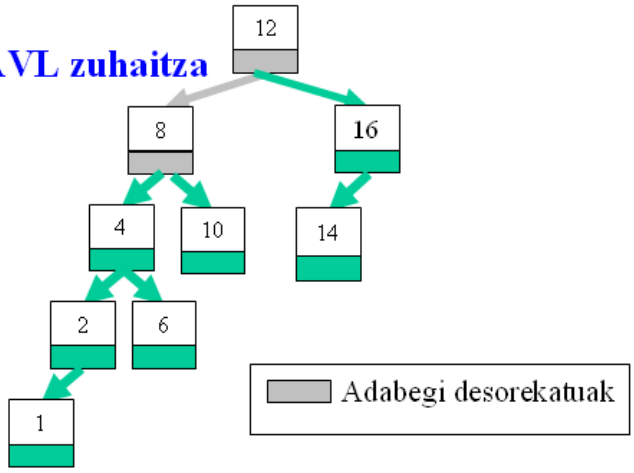
(Adelson-Velskii eta Landis):

- **Definizioak**:
  - bilaketa-zuhaitz bitarrak dira, baina orekazko ezaugarri osagarri batekin: **ezkerreko eta eskuineko umeen altueren aldea unitate batekoa izango da gehienez, adabegi guztientzat**

AVL zuhaitza

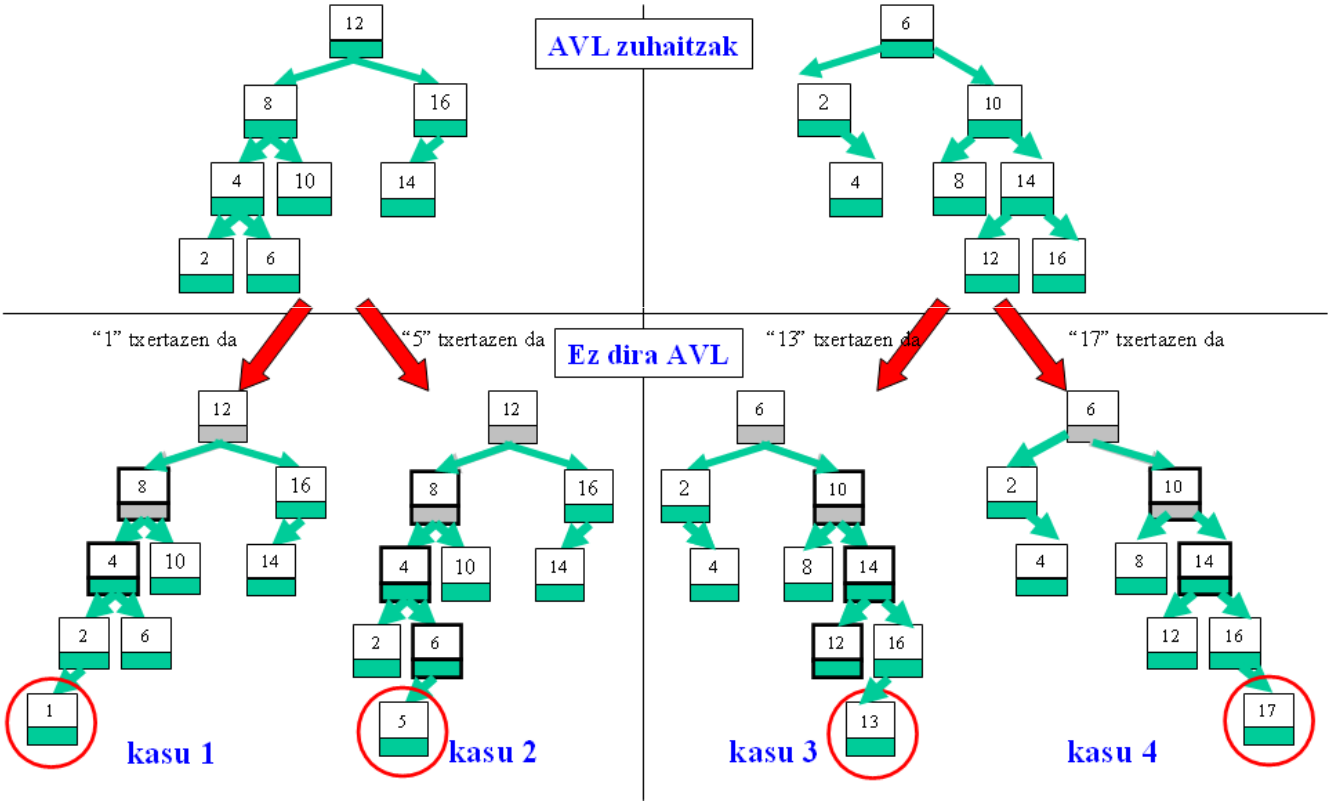


Ez da AVL zuhaitza



AVL zuhaitzak

4 posibilitate daude oreka hausteko txerkaketan:



AVL zuhaitzak

- AVL zuhaitza Bilaketa-zuhaitz bitarren inplementazio orekatua da
- Adabegi baten oreka-faktorea honi deituko diogu: eskuineko azpizuhaitzaren altuera ken ezkerreko azpizuhaitzaren altuera
- -1, 0 edo 1 oreka-faktorea duen adabegia zuhaitz orekatu baten erroa da
- AVL zuhaitz batean, adabegi guztien oreka-faktorea -1, 0 edo 1 da
- Bi era bakarrik daude zuhaitz orekatu bat desorekatzeko: adabegi bat gehitzean edo ezabatzean
- Horregatik, adabegi bat gehitu edo kentzen den bakoitzean, bere gaineko adabegien oreka-faktoreak aztertu beharko dira
- Así pues, cada vez que se añade o elimina un nodo se comprueban los factores de equilibrio de los nodos ascendientes del nodo en cuestión.
- AVL zuhaitz bateko adabegiek oreka-faktorea eta gurasoaren esteka bat izan dezakete, orekatze-eragiketak azkartu ahal izateko

- [Lewis eta Chase 2010]
- http://en.wikipedia.org/wiki/AVL_tree
- http://en.wikipedia.org/wiki/Tree_rotation

AVL zuhaitz baten orekatzea txertaketa baten ostean