

BILBOKO INGENIARITZA ESKOLA

KUDEAKETAREN ETA INFORMAZIO SISTEMEN
INFORMATIKAREN INGENIARITZAKO GRADUA

Aktoreak eta pelikulak kudeatu (4. eginkizuna)

Egileak:

Adei Arias

Jon Barbero

Ander Prieto

Arloa:

Datu-Egiturak eta

Algoritmoak

2. maila

46. taldea

1. lauhilabetea



Aurkibidea

1. Sarrera eta arazoaren aurkezpena	1
2. Diseinua	2
3. Datu egituren diseinua	3
4. Metodo nagusien diseinu eta inplementazioa	4
4.1. Aurreko eginkizunetako kodea	4
4.1.1. Datuak kargatu fitxategi batetik	4
4.1.2. Aktore baten bilaketa	5
4.1.3. Aktore berri baten txertaketa	5
4.1.4. Aktore baten pelikulak bueltatu	6
4.1.5. Pelikula bateko aktoreak bueltatu	6
4.1.6. Pelikula baten dirua gehitu	7
4.1.7. Aktore baten ezabaketa	7
4.1.8. Aktoreen zerrenda fitxategi batean gorde	8
4.1.9. Aktoreen zerrenda ordenatua lortu	9
4.1.10. Grafoa sortu	10
4.1.11. Konektatuta	10
4.1.12. Erlazionatuta	11
4.2. Laugarren eginkizuneko kodea	13
4.2.1. PageRank-a kalkulatu	13
4.2.2. PageRank-aren arabera lista ordenatua lortu	14

5. Kodea	16
5.1. Aurreko eginkizuneko kodea	16
5.1.1. Aktore.java	16
5.1.2. ArrayPelikulak.java	16
5.1.3. ArrayAktoreak.java	18
5.1.4. ListaAktoreak.java	20
5.1.5. ListaPelikula.java	24
5.1.6. Pelikula.java	26
5.2. Laugarren eginkizuneko kodea	27
5.2.1. GraphHash.java	27
5.2.2. Bikote.java	34
6. JUnitak	35
6.1. Aurreko eginkizuneko JUnitak	35
6.1.1. AktoreTest.java	35
6.1.2. ArrayPelikulakTest.java	37
6.1.3. ArrayAktoreakTest.java	39
6.1.4. ListaAktoreakTest.java	41
6.1.5. ListaPelikulaTest.java	44
6.1.6. PelikulaTest.java	46
6.2. Laugarren eginkizuneko JUnitak	49
6.2.1. GraphHashTest.java	49
7. Ondorioak	52

1. Sarrera eta arazoaren aurkezpena

Datu-Egiturak eta Algoritmoak ikasgaieko proiektua aktore eta pelikulen kudeaketa egitea da.

Ikasgai honetan, garrantzitsua da programaren kostua. Horretarako, hasiera-hasieratik azpimarratu dugu zer den kostua eta nola zeregin berdin baterako hainbat inplementazio ezberdin dagoen.

Beraz, hau argi ikusteko, hainbat eginkizun bete beharko ditugu lauhilabeteen zehar.

Laugarren eginkizun^[1] honetan, aktore eta pelikula bakoitzeko *PageRank* balioekin lan egingo dugu. *PageRank* bilaketa-tresna batek indexatutako dokumentuen (edo web orrialdeen) garrantziari balioa emateko erabiltzen den algoritmo sorta da.^[2]

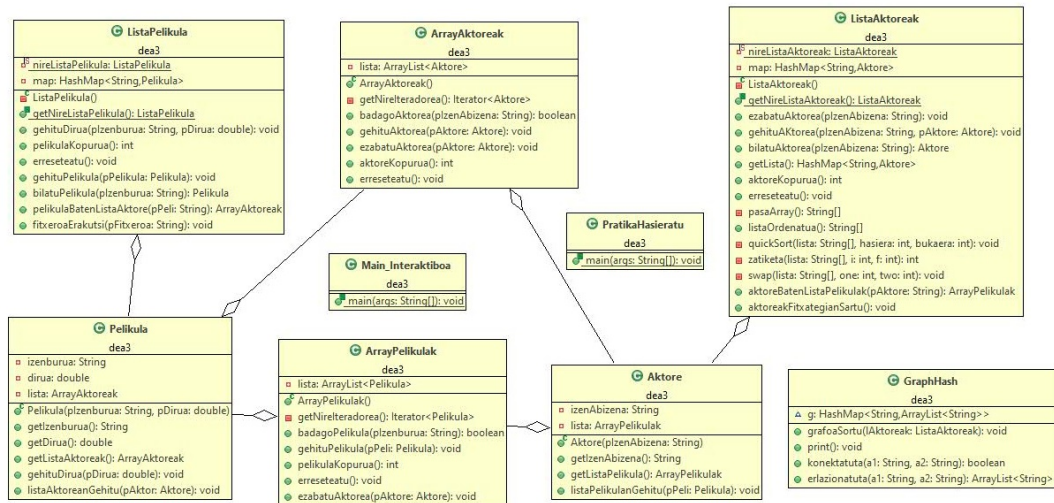
Eginkizun honetarako bi ataza ditugu. Hasteko, *PageRank* guztien kalkulua egitea da beharrezkoa. Ondoren, aktore edo pelikula baten izena emanda, bere pelikula edo aktore zerrenda, hurrenez hurren, inprimatuko du. Zerrenda hura ordenatuta egongo da, *PageRank*-aren arabera.

Gure praktikaren *main* metodoa *PraktikaHasieratu* klasean dago. Hala ere, interaktiboa den *main* bat jarri dugu (*Main_Interaktiboa* klasean), non lehen, hirugarren eta laugarren eginkizunen atazak dauden, egiazko kudeaketa programa bat balitz bezala.

Berriz ere, Eclipse gure “dantzarako bikote” izango dugu programatzeko momentuan, baita \LaTeX ere lortutako kodea, emaitzak eta hauen erreferentziak idazteko eta islatzeko.

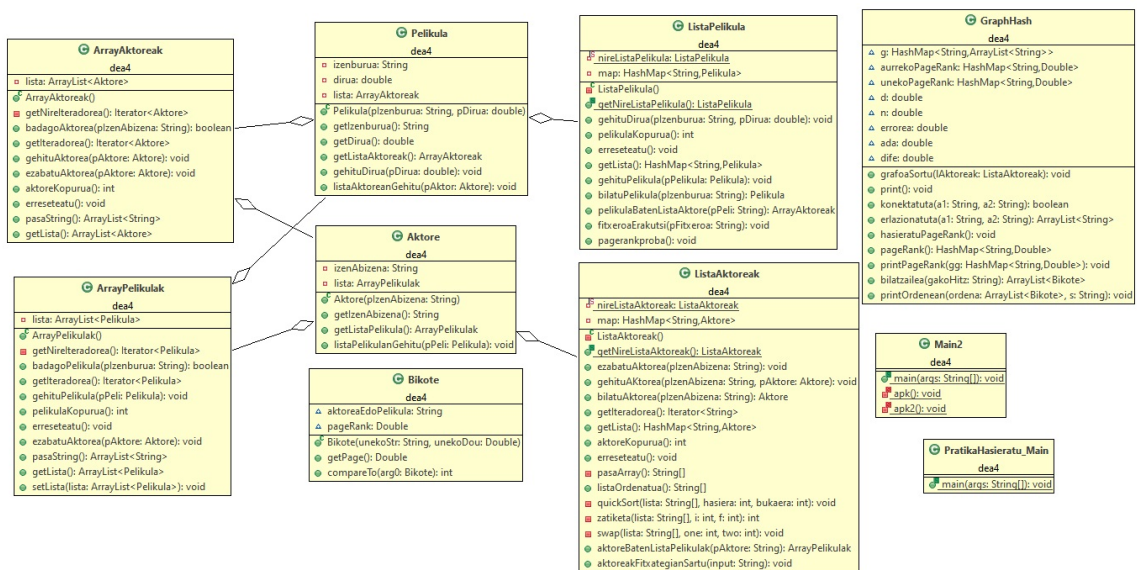
2. Diseinua

Laugarren eginkizun honetarako, hirugarren eginkizuneko diseinua oinarritzat hartu da (1. irudia).



1. irudia: Lehenengo eginkizuneko diseinua

Orain, aldiz, egin beharreko aldaketak txikiak dira; *GraphHash* izeneko klasean eginkizun honetako metodoak sortuko ditugu eta beharrezkoa den *Bikote* klasea sortuko dugu (2. irudia).



2. irudia: Diseinu berria

3. Datu egituren diseinua

Eginkizun honen diseinua hirugarren eginkizunaren antzkoa da. Bertan erabili genituen datu egiturak *HashMap*^[3], *HashSet*^[4], pilak eta ilarak dira.

Eginkizun honetan, aldiz, horietako batzuk eta beste berri batzuk erabili ditugu. Hasteko, *PageRank*-ak gordetzeko *HashMap* bi erabili ditugu: bata uneko iterazioko *PageRank*-en datuak gordetzeko eta bestea aurrekoko iteraziokoak gordetzeko.

Gainera, aktore edo pelikulen lista ordenatzeko *SortedSet*^[5] egitura erabili dugu; izan ere, egitura honetan elementuren bat sartzerakoan ordenean egiten du; barrutik *TreeSet* baten bezala portatzen denez, denbora logaritmikoan egiten du gehiketa.

4. Metodo nagusien diseinu eta implementazioa

4.1 AURREKO EGINKIZUNETAKO KODEA

4.1.1 *Datuak kargatu fitxategi batetik*

public void fitxeroaErakutsi(String pFitxeroa) throws FileNotFoundException, IOException

// Aurre: Fitxeroaren relative edo absolute path-a eman behar diogu.

// Post: Datuak programan kargatuko dira.

- Proba kasuak:

1. Datuak programan kargatzea
2. Fitxategia ez aurkitzea

- Algoritmoa:

```
sarrera = new Scanner(fitxategiaren izena)
while(sarrera.hasNext){
    lerroa=sarrera.nextLine();
    lerroa.split // pelikula | aktore guztiak batera
    pelikula
    pelik = new Pelikula(lerroa[0])
    gehituPelikula(pelik)
    aktoreak = lerroa[1].split // aktoreen Arraya
    while (aktoreak daude arrayan){
        aktor = bilatu aktorea HashMapean
        if (aktor==null){
            sartu aktorea
        }
        peli.gehitu(aktor) //pelikulari aktorea gehitu
        aktor.gehitu(peli) //aktoreari pelikula gehitu
    }
}
```

- Kostua: $O(n \cdot m)$ // n = lerroak iteratu; m = aktoreak iteratu

4.1.2 Aktore baten bilaketa

```
public Aktore bilatuAktorea(String pIzenAbizena)
```

```
// Aurre: Parametro gisa bilatu nahi den aktorearen izena eman behar da.
```

```
// Post: Aktorea aurkitzen badu, aktorea bera itzultzen du; bestela, null itzultzen du.
```

- Proba kasuak:

1. Aktorea badago.

- a) Elementu batez osatutako listan
- b) Elementu anitzez osatutako listan

2. Aktorea ez dago.

- a) Elementuz osatutako listan
- b) Lista hutsean

- Algoritmoa:

```
aktore=null;  
if (HashMap ListaAktoreak aktore badauka){  
    get aktore HashMapetik}  
return aktore;
```

- Kostua: $O(1)$

4.1.3 Aktore berri baten txertaketa

```
public void gehituAktorea(String pIzenAbizena, Aktore pAktore)
```

```
// Aurre: Izena eta aktorea bera pasatu behar zaio. Lehena HashMapean bilatzeko; egon  
ez badago, objektua sartzeko.
```

```
// Post: Aktorea txertatuko da.
```

- Proba kasuak:

1. Aktorea badago jada.
2. Aktorea ez dago oraindik.

- Algoritmoa:

```
if (HashMap ListaAktoreak pIzenAbizena EZ badauka){
    sartu pAktore HashMapean, pIzenAbizena gakoarekin}
```

- Kostua: $O(1)$

4.1.4 Aktore baten pelikulak bueltatu

// Post: Aktoreak fitxero berri batean sartuko dira. public ArrayPelikulak aktoreBaten-ListaPelikulak(String pAktore)

// Aurre: Aktore baten izena jasoko du parametro gisa.

// Post: Izen hori jakinda, HashMapean Aktore objektua bilatuko du eta honen lista bueltatuko du. Aurkitu gabe, null bueltatuko du.

- Proba kasuak:

1. listaAktorean dagoen aktore baten izena pasatzea.
2. listaAktorean ez dagoen aktore baten izena pasatzea.

- Algoritmoa:

```
Aktore aktor = this.bilatuAktorea(pAktore);
if(aktor == null)    return null;
else    return aktor.getListPelikula();
```

- Kostua: $O(1)$

4.1.5 Pelikula bateko aktoreak bueltatu

public ArrayAktoreak pelikulaBatenListaAktore(String pPeli)

// Aurre: Pelikula baten izena jasoko du parametro gisa.

// Post: Izenburu hori jakinda, HashMapean Pelikula objektua bilatuko du eta honen lista bueltatuko du. Aurkitu gabe, null bueltatuko du.

- Proba kasuak:

1. listaPelikulan dagoen pelikula baten izena pasatzea.
2. listaPelikulan ez dagoen pelikula baten izena pasatzea.

■ Algoritmoa:

```
Pelikula peli = this.bilatuPelikula(pPeli);  
if(peli == null) return null;  
else return peli.getListaAktoreak();
```

■ Kostua: $O(1)$

4.1.6 Pelikula baten dirua gehitu

```
public void gehituDirua(String pIzenburua, double pDirua)
```

```
// Aurre: Pelikularen izenburua eta diru kopurua sartu behar da.
```

```
// Post: Pelikula aurkitzean, dirua atxikituko zaio. Ez egotean, ez da ezer gertatuko
```

■ Proba kasuak:

1. Pelikula badago.
2. Pelikula ez dago.

■ Algoritmoa:

```
pelik Pelikula;  
if (HashMap ListaAktoreak pIzenburua badauka){  
    pelik=HashMapetik pelikula lortu //(this.map.get)  
    sartu pDirua pelik objektuan}
```

■ Kostua: $O(1)$

4.1.7 Aktore baten ezabaketa

```
public void ezabatuAktorea(String pIzenAbizena)
```

```
// Aurre: Izena pasatuko zaio parametro gisa.
```

```
// Post: Aktorea ezabatuko da; horretarako, bere pelikula bakoitzetik ere ezabatu beharko da. Aktorea ez balego, ez da ezer gertatuko.
```

- Proba kasuak:

1. Aktorea badago.
2. Aktorea ez dago.

- Algoritmoa:

```
Aktore aktor = bilatu pIzenAbizena HashMapean
ArrayPelikulak lista berria;
if(aktor!=null){
    lista = aktorearen ListaPelikula lortu
    lista.ezabatuAktorea metodoan, banan banan ezabatu aktorea
    ↪ pelikula guztietatik
}
```

- Kostua: $O(1)$

4.1.8 Aktoreen zerrenda fitxategi batean gorde

```
public void aktoreakFitxategianSartu()
```

```
// Aurre: Aktore guztien zerrenda hartuko da.
```

```
// Post: Aktoreak fitxero berri batean sartuko dira.
```

- Proba kasuak:

1. Fitxategia sortzea
2. Arazoa egotea fitxategia sortzean

- Algoritmoa:

```
fitxategia = new FileWriter(fitxategi berriaren izena)
Iterator it = aktoreen zerrenda lortu
while(it.hasNext){
    lerroa=it.next;
    if(it.hasNext){
        idatzi aktorea + "&&&" + lerro saltoa
    }
    else{
        idatzi lerro saltoa //hau soilik behin, hurrengo bueltan ez
        ↪ baita while begiztan sartuko
    }
}
```

```
    }
}
```

- Kostua: $O(n)$ // *While*-ak aktoreen *HashMap* guztia errekorritzen duelako

4.1.9 Aktoreen zerrenda ordenatua lortu

```
public String[] listaOrdenatua()
```

```
// Aurre: Lista ordenatu gabea izango dugu.
```

```
// Post: QuickSort metodoa erabiliz, lista ordenatuta lortuko da.
```

- Proba kasuak:

1. Lista ordenatua tratatu
2. Lista ez ordenatua tratatu
3. Lista hutsa tratatu

- Algoritmoa:

```
String lag = lista[i];
int ezker = i;
int eskuin = f;
while ( ezker < eskuin ){
    while ( lista[ezker].compareTo(lag) <= 0 && ezker < eskuin)
        ezker++;
    while ( lista[eskuin].compareTo(lag) > 0 )
        eskuin--;
    if ( ezker < eskuin )
        swap(lista, ezker, eskuin);
}
lista[i] = lista[eskuin];
lista[eskuin] = lag;
```

- Kostua: $O(n \log n)$

4.1.10 Grafoa sortu

```
public void grafoaSortu(ListaAktoreak lAktoreak)
```

```
// Aurre: --
```

```
// Post: Aktoreen zerrendatik grafoa sortzen du.
```

- Proba kasuak:

- Grafoa ondo kargatzea.
- Grafoa gaizki kargatzea.

- Algoritmoa:

```
aktore = null
pelikula = null
listaAktoreen iteradorea hasieratuko dugu
while(aktoreen lista guztia ez dugun iteratu)
    listaren hurrengo aktorea hartuko dugu iteradorearen bidez(.next())
    g.put(aktorearen izena, aktorearen listaPelikulak String-era pasatuz)
listaPelikulen iteradorea hasieratuko dugu
while(pelikulen lista guztia ez dugun iteratu)
    listaren hurrengo pelikula hartuko dugu iteradorearen bidez(.next())
    g.put(pelikularen izena, pelikularen listaAktoreak String-era pasatuz)
```

- Kostua: $O(n + m)$ // n : aktore kop., m : pelikula kop.

Egia da bai aktore eta bai pelikula listetan ArrayList batzuk iteratzen ditugula, baina hauen tamaina hain txikia denez bi lista nagusiekin konparatuta, algoritmoa kostu lineala izaten jarraitzen du.

4.1.11 Konektatuta

```
public boolean konektatuta(String a1, String a2)
```

```
// Aurre: Bi aktoreen izen-abizenak pasatuko dira, ".^bizena, izena"formatuan.
```

```
// Post: Bi aktore horien artean erlaziorik baldin badago esango du programak.
```

- Proba kasuak:

- Grafoan elementuak ez daude:

GraphHash	a1	a2	Emaitza
(a-b-c-d)	b	e	False; "e ez dago grafoan"
(a-b-c-d)	e	a	False; "e ez dago grafoan"
(a-b-c-d)	e	f	False; "e eta f ez daude grafoan"

- Grafoan elementuak badaude:

GraphHash	a1	a2	Emaitza
(a-b-c-d) (e-f-g-h)	a	d	True
(a-b-c-d) (e-f-g-h)	a	f	False

- Algoritmoa:

```

Queue<String> aztGabe = new LinkedList<String>()
HashSet<String> aztertuak = new HashSet<String>()
konek = false
"a1" sartu aztGabe ilaran
"a1" sartu HashSet-ean
Parametroen bidez pasatutako aktoreak ez baleude grafoan
mezuak erakutsi, agertzen ez den aktorea zein den esanez
while(aztGabe ez den hutsa eta konek false den)
    eg = aztGabe ilaratik lehenengo elementua kendu
    if(eg.equals(a2))
        konek = true
    else
        ArrayList<String> array = eg-ren lista lortu
        array-ren iteradorea lortu
        while(array ez den bukatu)
            izena = array-ren hurrengo elementua
            if(izena HashSet-ean ez badago)
                ilaran sartu izena
                HashSet-ean sartu izena
konek itzuli

```

- Kostua: $O(n)$ // n : grafoko elementu guztiak

Grafoa sortzean gertatu den arazo berdinarekin aurrean gaude: lehenengo while-aren barruan, HashMap-eko elementu bakoitzaren ArrayList-a iteratzen dugu, baina hau hain txikia denez "nrekin konparatuz, kostea linea izaten jarraitzen du.

4.1.12 Erlazionatuta

```
public ArrayList<String>erlazionatuta(String a1, String a2)
```

// Aurre: Bi aktoreen izen-abizenak pasatuko dira, “abizena, izena” formatuan.

// Post: Bi aktore horien artean erlaziorik baldin badago, erlazionatzen duen bidea itzuliko du. Erlaziorik ez balego, aldiz, null.

■ Proba kasuak:

- Grafoan elementuak ez daude:

GraphHash	a1	a2	Eraitza
(a-b-c-d)	b	e	null; “e ez dago grafoan”
(a-b-c-d)	e	a	null; “e ez dago grafoan”
(a-b-c-d)	e	f	null; “e eta f ez daude grafoan”

- Grafoan elementuak badaude:

GraphHash	a1	a2	Eraitza
(a-b-c-d-e) (f-g-h)	a	e	<a>,,<c>,<d>,<e>
(a-b-c-d-e) (f-g-h)	a	f	null; “Aktoreak ez daude konektatuta”

■ Algoritmoa:

```

aux=konektatuta(a1,a2);
atera=false;
Queue<String> aztGabe = new LinkedList<String>();
HashSet<String> aztertuak = new HashSet<String>();
Stack<String> nondik = new Stack<String>();
Stack<String> adabegia = new Stack<String>();
ArrayList<String> eraitza=new ArrayList<String>();
"a1" sartu aztGabe ilaran
"a1" sartu HashSet-ean
"a1" sartu adabegia pila
while(aztGabe ez den hutsa eta konek false den)
    eg = aztGabe ilaratik lehenengo elementua kendu
    if(eg.equals(a2))
        konek = true
    else
        ArrayList<String> array = eg-ren lista lortu
        array-ren iteradorea lortu
        while(array ez den bukatu)
            izena = array-ren hurrengo elementua
            if(izena HashSet-ean ez badago)
                ilaran sartu izena
                HashSet-ean sartu izena

```



```

        nondik pilan sartu eg
        adabegia pilan sartu izena
    aurrekoa = a2
    sartu ArrayList emaitzan a2
    while(adabegia pila ez den hutsa eta aurrekoa ez den a1)
        while(adabegia-ren lehenengo elementua ez den aurrekoa eta nondik ez den hu
            nondik pilatik elementu bat kendu
            adabegia pilatik elementua bat kendu
            aurrekoa = nondik pilatik elementu bat kenduta
            aurrekoa gorde emaitza arrayListean
            adabegia pilatik elementu bat kendu
    else
        mezu bat atera, aktoreak ez daudela konektatuta esanez
    emaitza arraylista bueltatu

```

- Kostua: $O(n + m + p)$ // n : grafoko elementu kop., m : adabegi pilaren elementu kop, p : emaitzeko ArrayList-aren elementu kop.

Hemen, bi zati bereizten dira. Lehenengo zatia, n -rena, konektatu metodoaren kostu berdina izango du; hau da, lineala izango da. Eta bigarren zatia, lehenengo zatiarekiko guztiz independentea dena, berriro ere kostu lineala izango du. Azken batean, bigarren zati honetan, ArrayList-a betetzen joango gara, eta honetarako bi while erabiliko ditugu, bat bestearen barruan. Baina kostua ez da koadratikoa izango; izan ere, “adabegia” izeneko pila bat husten joango gara bi while-tan. Horregatik, kostua lineala izaten jarraituko du. Bi zatiak independenteak direnez, kostuak gehituko dira.

4.2 LAUGARREN EGINKIZUNEN KODEA

4.2.1 PageRank-a kalkulatu

```

public HashMap<String, Double>pageRank()

// Aurre: pagerankHasieratu() metodoa exekutatu da aurretik.

// Post: Grafoaren elementu bakoitzak bere PageRank balioa izango du.

```

- Proba kasuak:
 - Errorerik ez dago; grafoa ondo itzuliko du.
 - Errorrea badago; baliteke gaizki hasieratuta egotea.

- Algoritmoa:

```
double diferentzia
while (diferentzia>errorea){
    diferentzia=0.0
    grafoaren iteradorea lortu
    while (itr.hasNext){
        elem = lortu itr.next
        elemen lista lortu
        for (listaren luzera){
            lortu listako i elementua
            lortu listaren luzera
            lortu aurreko iterazioko pagerank balioa
            lortu uneko iterazioko pagerank balioa
            uneko pagerank balioa eguneratu
        }
    }
    uneko pagerank grafoaren iteradorea lortu
    while (itr.hasNext){
        i elementuari uneko pagerank balioari formula aplikatu
        i elementua aurreko pagerank grafoan 0.0-ra jarri
    }
    aurreko pagerank eta uneko pagerank grafoak elkartrukatu
}
return aurreko pagerank grafoa
```

- Kostua: $O(n + m)$ // n : grafoko elementu kop., m : elementu bakoitzaren listaren elementu kop.

4.2.2 PageRank-aren arabera lista ordenatua lortu

```
public ArrayList<Bikote>bilatzailea(String gakoHitz)
```

```
// Aurre: pageRank() metodoa exekutatu da aurretik.
```

// Post: Emaizta emandako gako-hitzarekin lotuta dauden elementuen zerrenda da (aktoreak edo pelikulak, gakoaren arabera), bere PageRank-aren arabera handienetik txikienera ordenatuta (hau da, lehenengo posizioetan PageRank handiena duten elementuak agertuko dira).

- Proba kasuak:

- Errorerik ez dago; lista ordenean itzuliko du.
- Errorea badago; baliteke gaizki hasieratuta egotea.

■ Algoritmoa:

```
ArrayList<Bikote> emaitza = new ArrayList<Bikote>()
ArrayList<String> al = lortu gakohitzen lista
SortedSet<Bikote> set = new TreeSet<Bikote>()
for (al-ren luzera){
    unekostr = al-ren i elementua lortu
    unekodou = i elementuaren pagerank balioa lortu
    set.add(new Bikote(unekostr, unekodou))
}
set-ren iteradorea lortu
while (itr.hasNext){
    bik = itr.next
    emaitza.add(bik)
}
return emaitza
```

- Kostua: $O(n \log n)$ // n : listako elementu kop.

5. Kodea

5.1 AURREKO EGINKIZUNEKO KODEA

5.1.1 *Aktore.java*

```
1 package dea3;
2
3 public class Aktore {
4
5     private String izenAbizena;
6     private ArrayPelikulak lista;
7
8     public Aktore(String pIzenAbizena){
9         this.izenAbizena = pIzenAbizena;
10        this.lista = new ArrayPelikulak();
11    }
12
13    public String getIzenAbizena(){
14        return this.izenAbizena;
15    }
16
17    public ArrayPelikulak getListaPelikula(){
18        return this.lista;
19    }
20
21    public void listaPelikulanGehitu(Pelikula pPeli){//Hemen, pelikula bat sartuko
22        ↪ dugu aktorearen listaPelikulan
23        this.lista.gehituPelikula(pPeli);
24    }
```

5.1.2 *ArrayPelikulak.java*

```
1 package dea3;
2
3 import java.util.ArrayList;
4 import java.util.Iterator;
5
6 public class ArrayPelikulak {
7
8     private ArrayList<Pelikula> lista;
9
10    public ArrayPelikulak(){
11        this.lista = new ArrayList<Pelikula>();
12    }
13
14    private Iterator<Pelikula> getNireIteradorea(){
15        return this.lista.iterator();
16    }
17
18    public boolean badagoPelikula(String pIzenburua){//Pelikula bat emanda, listan
19        ↪ dagoen esango digu
20        boolean dago = false;
21        Pelikula pelikula = null;
22        Iterator<Pelikula> itr = this.getNireIteradorea();
23        while(itr.hasNext() && !dago){
24            pelikula = itr.next();
25            if(pelikula.getIzenburua().equals(pIzenburua)){
26                dago = true;
27            }
28        }
29        return dago;
30    }
31
32    public Iterator<Pelikula> getIteradorea(){
33        return this.lista.iterator();
34    }
35
36    public void gehituPelikula(Pelikula pPeli){
37        this.lista.add(pPeli);
38    }
39
40    public int pelikulaKopurua() {
41        return this.lista.size();
42    }
43
44    public void erreseteatu() {
45        this.lista.clear();
46    }
47 }
```

```
45     }
46
47     public void ezabatuAktorea(Aktore pAktore) {
48         Pelikula peli = null;
49         ArrayAktoreak lista=null;
50         Iterator<Pelikula> itr = this.getNireIteradorea();
51         while(itr.hasNext()) {
52             peli = itr.next();
53             lista=peli.getListaAktoreak();
54             lista.ezabatuAktorea(pAktore);
55         }
56     }
57
58     public ArrayList<String> pasaString(){
59         Iterator<Pelikula> itr = this.getNireIteradorea();
60         Pelikula peli = null;
61         ArrayList<String> lista = new ArrayList<String>();
62         while(itr.hasNext()){
63             peli = itr.next();
64             lista.add(peli.getIzenburua());
65         }
66         return lista;
67     }
68 }
```

5.1.3 ArrayAktoreak.java

```
1 package dea3;
2
3 import java.util.*;
4
5 public class ArrayAktoreak {
6
7     private ArrayList<Aktore> lista;
8
9     public ArrayAktoreak(){
10         this.lista = new ArrayList<Aktore>();
11     }
12 }
```

```

12
13 private Iterator<Aktore> getNireIteradorea(){
14     return this.lista.iterator();
15 }
16
17 public boolean badagoAktorea(String pIzenAbizena){//Aktore bat pasata, listan
18     ↪ dagoen esango digu
19     boolean dago = false;
20     Aktore aktor = null;
21     Iterator<Aktore> itr = this.getNireIteradorea();
22     while(itr.hasNext() && !dago){
23         aktor = itr.next();
24         if(aktor.getIzenAbizena().equals(pIzenAbizena)){
25             dago = true;
26         }
27     }
28     return dago;
29 }
30 public Iterator <Aktore> getIteradorea(){
31     return this.lista.iterator();
32 }
33
34 public void gehituAktorea(Aktore pAktore){
35     if(!(this.badagoAktorea(pAktore.getIzenAbizena()))){
36         this.lista.add(pAktore);
37     }
38 }
39
40 public void ezabatuAktorea(Aktore pAktore) {
41     this.lista.remove(pAktore);
42 }
43
44 public int aktoreKopurua() {
45     return this.lista.size();
46 }
47
48 public void erreseteatu() {
49     this.lista.clear();
50 }
51
52 public ArrayList<String> pasaString(){
53     Iterator<Aktore> itr = this.getNireIteradorea();
54     Aktore aktore = null;
55     ArrayList<String> lista = new ArrayList<String>();
56     while(itr.hasNext()){
57         aktore = itr.next();

```

```
57     lista.add(aktore.getIzenAbizena());
58 }
59 return lista;
60 }
61 }
```

5.1.4 *ListaAktoreak.java*

```
1 package dea3;
2
3 import java.util.*;
4 import java.io.FileNotFoundException;
5 import java.io.FileWriter;
6 import java.io.IOException;
7
8 public class ListaAktoreak {
9
10     private static ListaAktoreak nireListaAktoreak = null;
11     private HashMap<String,Aktore> map;
12
13
14     private ListaAktoreak(){
15         this.map = new HashMap<String,Aktore>();
16     }
17
18     public static ListaAktoreak getNireListaAktoreak(){
19         if(nireListaAktoreak == null){
20             nireListaAktoreak = new ListaAktoreak();
21         }
22         return nireListaAktoreak;
23     }
24
25     public void ezabatuAktorea(String pIzenAbizena){
26         Aktore aktor=this.bilatuAktorea(pIzenAbizena);
27         ArrayPelikulak lista=null;
28         if(aktor!=null){
29             lista=aktor.getListaPelikula();
30             lista.ezabatuAktorea(aktor);
```



```
31     this.map.remove(pIzenAbizena);
32 }
33 }
34
35 public void gehituAktorea(String pIzenAbizena, Aktore pAktore){
36     if(!this.map.containsKey(pIzenAbizena)){
37         this.map.put(pIzenAbizena, pAktore);
38     }
39 }
40
41 public Aktore bilatuAktorea(String pIzenAbizena){
42     Aktore aktor = null;
43     if(this.map.containsKey(pIzenAbizena)){
44         aktor = this.map.get(pIzenAbizena);
45     }
46     return aktor;
47 }
48
49 public Iterator<String> getIteradorea(){
50     return this.map.keySet().iterator();
51 }
52
53 public HashMap<String,Aktore> getLista(){
54     return this.map;
55 }
56
57 public int aktoreKopurua() {
58     return this.map.size();
59 }
60
61 public void erreseteatu() {
62     this.map.clear();
63 }
64
65
66 private String[] pasaArray(){
67     String[] lista = new String[this.aktoreKopurua()];
68     int i = 0;
69     Iterator<String> it = map.keySet().iterator();
70     String izena = null;
71     while (it.hasNext()){
72         izena = it.next();
73         lista[i]=izena;
74         i=i+1;
75     }
76     return lista;
```

```
77     }
78
79     public String[] listaOrdenatua(){
80         String[] lista = this.pasaArray();
81         quickSort(lista, 0, lista.length-1);
82         return lista;
83     }
84
85
86     private void quickSort(String[] lista, int hasiera, int bukaera){
87         if ( bukaera - hasiera > 0 ) { // taulan elementu bat baino gehiago
88             int indizeaZatiketa = zatiketa(lista, hasiera, bukaera);
89             quickSort(lista, hasiera, indizeaZatiketa - 1);
90             quickSort(lista, indizeaZatiketa + 1, bukaera);
91         }
92     }
93
94     private int zatiketa(String[] lista, int i, int f){
95
96         String lag = lista[i];
97         int ezker = i;
98         int eskuin = f;
99         while ( ezker < eskuin ){
100             lag.toUpperCase();
101             lista[ezker].toUpperCase();
102             lista[eskuin].toUpperCase();
103             while ( lista[ezker].compareTo(lag) <= 0 && ezker < eskuin)
104                 ezker++;
105             while ( lista[eskuin].compareTo(lag) > 0 )
106                 eskuin--;
107             if ( ezker < eskuin )
108                 swap(lista, ezker, eskuin);
109         }
110         lista[i] = lista[eskuin];
111         lista[eskuin] = lag;
112
113         return eskuin;
114     }
115
116     private void swap(String[] lista, int one, int two) {
117         String temp = lista[one];
118         lista[one] = lista[two];
119         lista[two] = temp;
120     }
121
122
```

```
123 public ArrayPelikulak aktoreBatenListaPelikulak(String pAktore){
124     Aktore aktor = this.bilatuAktorea(pAktore);
125     if(aktor == null){
126         return null;
127     }else{
128         return aktor.getListaPelikula();
129     }
130 }
131
132 public void aktoreakFitxategianSartu(){
133     FileWriter fitxategia1 = null;
134     try {
135
136         fitxategia1 = new FileWriter("./FilmsActors20162017Fitxategia.txt");
137         Iterator<String> it =
138             ↳ ListaAktoreak.getNireListaAktoreak().getList().keySet().iterator();
139         String lerroa = null;
140         // Lerro bakoitza fitxategian idazten dugu
141         while (it.hasNext()) {
142             lerroa = it.next();
143             if(it.hasNext()){
144                 fitxategia1.write(lerroa + " &&& " + "\n");
145             }else{
146                 fitxategia1.write(lerroa + "\n");
147             }
148
149             fitxategia1.close();
150
151         }
152         catch (FileNotFoundException e) {
153             System.out.println("Fitxeroa ez da existitzen. ");
154         } catch (IOException e) {
155             System.out.println("Fitxategiaren idazketak huts egin du. ");
156         }
157     }
158 }
```

5.1.5 *ListaPelikula.java*

```
1 package dea3;
2
3 import java.io.BufferedReader;
4 import java.io.FileInputStream;
5 import java.io.FileNotFoundException;
6 import java.io.FileReader;
7 import java.io.IOException;
8 import java.io.InputStreamReader;
9 import java.util.*;
10
11 public class ListaPelikula {
12
13     private static ListaPelikula nireListaPelikula = null;
14     private HashMap<String,Pelikula> map;
15
16     private ListaPelikula(){
17         this.map = new HashMap<String,Pelikula>();
18     }
19
20     public static ListaPelikula getNireListaPelikula(){
21         if(nireListaPelikula == null){
22             nireListaPelikula = new ListaPelikula();
23         }
24         return nireListaPelikula;
25     }
26
27     public void gehituDirua(String pIzenburua, double pDirua){
28         Pelikula pelik = null;
29         if(this.map.containsKey(pIzenburua)){
30             pelik=this.map.get(pIzenburua);
31             pelik.gehituDirua(pDirua);
32         }
33     }
34
35     public int pelikulaKopurua() {
36         return this.map.size();
37     }
38
39     public void erreseteatu() {
40         this.map.clear();
41     }
42 }
```

```

43 public HashMap<String,Pelikula> getLista(){
44     return this.map;
45 }
46
47 public void gehituPelikula(Pelikula pPelikula){
48     if(this.bilatuPelikula(pPelikula.getIzenburua())==null){
49         this.map.put(pPelikula.getIzenburua(),pPelikula);
50     }
51 }
52
53 public Pelikula bilatuPelikula(String pIzenburua){
54     Pelikula pelikula = null;
55     if (this.map.containsKey(pIzenburua)){
56         pelikula=this.map.get(pIzenburua);
57     }
58     return pelikula;
59 }
60
61 public ArrayAktoreak pelikulaBatenListaAktore(String pPeli){
62     Pelikula peli = this.bilatuPelikula(pPeli);
63     if(peli == null){
64         return null;
65     }else{
66         return peli.getListAktoreak();
67     }
68 }
69
70 public void fitxeroaErakutsi(String pFitxeroa) throws FileNotFoundException,
↪ IOException{
71     try{
72         Scanner entrada = new Scanner(new FileReader(pFitxeroa));
73         String linea;
74         Aktore aktor;
75         Pelikula peli = null;
76         while (entrada.hasNext()) {
77             linea = entrada.nextLine();
78             String[] datuak = linea.split("\\s+--->\\s+");
79             peli = new Pelikula(datuak[0],45.00);
80             ListaPelikula.getNireListaPelikula().gehituPelikula(peli);
81             //System.out.println(datuak[0]);
82             String[] aktoreak = datuak[1].split("\\s+&&\\s+");
83             int i=0;
84             while (i < aktoreak.length){//sartu aktoreak eta pelikulak
85                 aktor =
↪ ListaAktoreak.getNireListaAktoreak().bilatuAktorea(aktoreak[i]);
86                 if(aktor==null){

```

```

87         aktor = new Aktore(aktoreak[i]);
88         ListaAktoreak.getNireListaAktoreak().gehituAktorea(aktoreak[i],
            ↳ aktor);
89     }
90     //System.out.println(aktoreak[i]);
91     peli.listaAktoreanGehitu(aktor); //Hemen, pelikulari aktore hau
            ↳ sartuko diogu bere listaAktorean
92     aktor.listaPelikulanGehitu(peli); //Hemen, aktore honi sartuko
            ↳ diogu pelikula hau bere listaPelikulan
93     i++;
94 }
95 }
96 entrada.close();
97 } catch (IOException e) {e.printStackTrace();}
98 }
99 }

```

5.1.6 Pelikula.java

```

1 package dea3;
2
3 public class Pelikula {
4
5     private String izenburua;
6     private double dirua;
7     private ArrayAktoreak lista;
8
9     public Pelikula(String pIzenburua, double pDirua){
10         this.izenburua = pIzenburua;
11         this.dirua = pDirua;
12         this.lista = new ArrayAktoreak();
13     }
14
15     public String getIzenburua(){
16         return this.izenburua;
17     }
18
19     public double getDirua(){

```

```

20     return this.dirua;
21 }
22
23 public ArrayAktoreak getListAktoreak(){
24     return this.lista;
25 }
26
27 public void gehituDirua(double pDirua){
28     this.dirua = this.dirua + pDirua;
29 }
30 public void listaAktoreanGehitu(Aktore pAktor){//Hemen, aktore bat sartuko dugu
31     ↪ pelikularen listaAktorean
32     this.lista.gehituAktorea(pAktor);
33 }

```

5.2 LAUGARREN EGINKIZUNEN KODEA

5.2.1 *GraphHash.java*

```

1 package dea4;
2
3 import java.text.DecimalFormat;
4 import java.text.DecimalFormatSymbols;
5 import java.util.ArrayList;
6 import java.util.HashMap;
7 import java.util.HashSet;
8 import java.util.Iterator;
9 import java.util.LinkedList;
10 import java.util.Locale;
11 import java.util.Queue;
12 import java.util.SortedSet;
13 import java.util.Stack;
14 import java.util.TreeSet;
15
16 public class GraphHash {
17     HashMap<String, ArrayList<String>> g = new HashMap<String, ArrayList<String>>();
18 }

```

```

19
20 HashMap<String,Double> aurrekoPageRank = new HashMap<String,Double>();//elementu
    ↳ bakoitzaren pagerank-a sartzeko lista
21 HashMap<String,Double> unekoPageRank = new HashMap<String,Double>();//elementu
    ↳ bakoitzaren pagerank-a sartzeko lista
22
23 double d = 0.85;
24 double n = 1/((double)g.size());
25 double errorea = 0.0001;
26 double ada=0.0;
27 double dife=0.0;
28
29
30 public void grafoaSortu(ListaAktoreak lAktoreak){
31     // Post: aktoreen zerrendatik grafoa sortzen du
32     // Adabegiak aktoreen izenak eta pelikulen izenburuak dira
33     // KODEA OSATU
34     Aktore aktor = null;
35     Pelikula peli = null;
36     Iterator<Aktore> itrAktor = lAktoreak.getListA().values().iterator();
37     //Lehenengo akoreak eta ondoren pelikulak sartuko ditugu hashmap-ean
38     while(itrAktor.hasNext()){
39         aktor = itrAktor.next();
40         String izenaAktor = aktor.getIzenAbizena();
41         g.put(izenaAktor, aktor.getListaPelikula().pasaString());
42         //pasaString() metodoa erabili dugu, arrayList-a(bai aktoreak gordetzen
            ↳ dituena eta bai pelikulak gordetzen dituen)
43         //string-era pasatzeko eta hashmapean sartzeko
44     }
45     Iterator<Pelikula> itrPeli =
        ↳ ListaPelikula.getNireListaPelikula().getListA().values().iterator();
46     while(itrPeli.hasNext()){
47         peli = itrPeli.next();
48         String izenaPeli = peli.getIzenburua();
49         g.put(izenaPeli, peli.getListAktoreak().pasaString());
50     }
51 }
52 public void print(){
53     int i = 1;
54     for (String s: g.keySet()){
55         System.out.print("Element: " + i++ + " " + s + " --> ");
56         for (String k: g.get(s)){
57             System.out.print(k + " ### ");
58         } System.out.println();
59     }
60 }

```



```

61
62 public boolean konektatuta(String a1, String a2){
63     Queue<String> aztGabe = new LinkedList<String>();
64     HashSet<String> aztertuak = new HashSet<String>();
65     aztGabe.add(a1);
66     aztertuak.add(a1);
67     //lehenengo elementua sartuko dugu, geroago agertzen bada, berriro ere ez
        ↪ sartzeko
68     boolean konek = false;
69     if(ListaAktoreak.getNireListaAktoreak().bilatuAktorea(a1)==null &&
        ↪ ListaAktoreak.getNireListaAktoreak().bilatuAktorea(a2)==null){
70         System.out.println("Sartutako bi aktoreak ez daude grafoan sartuta");
71     }else if(ListaAktoreak.getNireListaAktoreak().bilatuAktorea(a1)==null){
72         System.out.println("Sartutako lehenengo aktorea ez dago grafoan sartuta");
73     }else if(ListaAktoreak.getNireListaAktoreak().bilatuAktorea(a2)==null){
74         System.out.println("Sartutako bigarren aktorea ez dago grafoan sartuta");
75     }else{
76         while(!aztGabe.isEmpty() && !konek){
77             String eg= aztGabe.remove();
78             if(eg.equals(a2)){
79                 konek=true;
80             }else{
81                 ArrayList<String> array = g.get(eg);
82                 Iterator<String> itr = array.iterator();
83                 while(itr.hasNext()){//ilaratik hartutako elementuaren arrayList-a
                    ↪ iteratuko dugu honen elementuak pila sartzeko
84                     String izena = itr.next();
85                     if(!aztertuak.contains(izena)){
86                         aztGabe.add(izena);
87                         aztertuak.add(izena);
88                         //bai ilaran eta bai hashSet-ean sartuko ditugu, bi datu
                            ↪ egituretan elementuak ez errepikatzeko
89                     }
90                 }
91             }
92         }
93     }
94     return konek;
95 }
96
97 public ArrayList<String> erlazionatuta(String a1, String a2){
98     boolean aux=konektatuta(a1,a2);
99     boolean atera=false;
100     Queue<String> aztGabe = new LinkedList<String>();
101     HashSet<String> aztertuak = new HashSet<String>();
102     Stack<String> nondik = new Stack<String>();//pila honen helburua, grafoaren
        ↪ elementu bakoitza nondik etorri den jakitea da

```

```

103 Stack<String> adabegia = new Stack<String>();//pila honetan, grafoaren elementu
    ↪ guztiak sartuko ditugu
104 aztGabe.add(a1);
105 aztertuak.add(a1);
106 adabegia.add(a1);
107 //lehenengo elementua sartuko dugu, geroago aterako balitz, berriro ez sartzeko
108 ArrayList<String> emaitza=new ArrayList<String>();
109 if(aux){//bi aktoreak konektatuta badaude sartuko da
110     while(!aztGabe.isEmpty() && !atera){
111         String lag= aztGabe.remove();
112         if(lag.equals(a2)){
113             atera=true;
114         }else{
115             ArrayList<String> array = g.get(lag);
116             Iterator<String> itr = array.iterator();
117             while(itr.hasNext()){
118                 String izena = itr.next();
119                 if(!aztertuak.contains(izena)){
120                     aztGabe.add(izena);
121                     aztertuak.add(izena);
122                     nondik.add(lag);//hemen, lehen esan den bezala, beste datu
    ↪ egituretan sartzen ari garen elementuaren
123 //gurasoak sartuko ditugu
124                     adabegia.add(izena);
125                     //aurreko metodoan egin den bezala, datu egitura guztietan
    ↪ sartuko da, berriro ere ateratzen bada
126 //elementu berdina, ez sartzeko
127                 }
128             }
129         }
130     }//ARRAYLIST-A BETETZEN HASI
131 String aurrekoa=a2;//Lehenengo elementua sartuko dugu pilan
132 Stack<String> emaitza2 = new Stack<String>();
133 emaitza2.push(a2);
134 while(!adabegia.isEmpty() && !aurrekoa.equals(a1)){
135     while(!adabegia.peek().equals(aurrekoa) && !nondik.isEmpty()){
136         //while honetan, aurrekoa atributuan daukagun balioa aurkitu beharko
    ↪ dugu adabegia pilan, eta hau egiten dugun
137 //bitartean, nondik pilan elementuak ateratzen joango gara.
138 //Elementua aurkitzean, nondik pilan dagoen azken String-a,
    ↪ elementuaren gurasoa izango da, eta hau pilan
139 //sartuko dugu. Nondik, hutsa denean, azken-aurreko elementura iritsi
    ↪ gara(azken elementua metodoaren parametroko
140 //string bat da)
141     nondik.pop();
142     adabegia.pop();

```

```
143     }
144     aurrekoa = nondik.pop();
145     emaitza2.push(aurrekoa);
146     adabegia.pop();
147 }
148 //momentu honetan, bi aktoreen arteko erlazioaren bidea pilan dago, eta
149 ↪ orain arrayListera pasako dugu
150 while(!emaitza2.isEmpty()){
151     emaitza.add(emaitza2.pop());
152 }
153 }
154 else{
155     System.out.println("Aktoreak ez daude konektatuta");
156 }
157 return emaitza;
158 }
159
160 public void hasieratuPageRank(){
161     n= 1.0/((double)g.size());
162     double balioa=n;
163     String elem="";
164     Iterator<String> itr= g.keySet().iterator();
165     while(itr.hasNext()){
166         elem = itr.next();
167         aurrekoPageRank.put(elem, balioa);
168         unekoPageRank.put(elem, 0.0);
169     }
170 }
171
172 public HashMap<String, Double> pageRank(){
173     DecimalFormat df = new DecimalFormat("0",
174 ↪ DecimalFormatSymbols.getInstance(Locale.ENGLISH));
175     df.setMaximumFractionDigits(340);
176     double diferentzia=9.9;
177     int j=1;
178     while(diferentzia>errorea){
179         diferentzia=0.0;
180
181         Iterator<String> itr = g.keySet().iterator();
182
183         while(itr.hasNext()){
184             String elem= itr.next();
185             int listaLuzera = g.get(elem).size();
186             for(int i=0;i<listaLuzera ; i++){
187                 String listakoElem = g.get(elem).get(i);
```

```

187         Double listakoElemListaLuzera = (double)g.get(listakoElem).size();
188         Double listakoElemPR = aurrekoPageRank.get(listakoElem);
189         //System.out.println(listakoElemPR);
190         Double elemPR = unekoPageRank.get(elem);
191         //System.out.println(elemPR);
192         unekoPageRank.put(elem,elemPR+(listakoElemPR/listakoElemListaLuzera));
193         unekoPageRank.get(elem);
194     }
195 }
196
197 Iterator<String> itrMap = unekoPageRank.keySet().iterator();
198
199 while (itrMap.hasNext()) {
200     String elemMap = itrMap.next();
201     double val1=unekoPageRank.get(elemMap);
202     double pr= ((1-d)/(1/n)) + (d*val1);
203     unekoPageRank.put(elemMap, pr);
204
205     double val2=aurrekoPageRank.get(elemMap);
206
207     double dif=Math.abs(pr-val2);
208     diferentzia=diferentzia+dif;
209     aurrekoPageRank.put(elemMap, 0.0);
210 }
211
212 HashMap<String, Double> temp = aurrekoPageRank;
213 aurrekoPageRank=unekoPageRank;
214 unekoPageRank=temp;
215 System.out.println(j+". iterazioa bukatuta, diferentzia " +
216     ↪ df.format(diferentzia) + " da (errorea = 0.0001)");
217 j++;
218 }
219 dife=diferentzia;
220 return aurrekoPageRank;
221 }
222
223 public void printPageRank(HashMap<String,Double> gg){
224     int i = 1;
225     System.out.println("PAGERANK LISTA:");
226     for (String s: gg.keySet()){
227         System.out.println(i++ + ". elementua: " + s + "; PageRank balioa: " +
228             ↪ gg.get(s));
229     }
230     System.out.println("Diferentzia finala: " + dife);
231 }

```

```
231
232 public ArrayList<Bikote> bilatzailea(String gakoHitz){
233     ArrayList<String> al = this.g.get(gakoHitz);
234     if (al==null) return null;
235     else{
236         ArrayList<Bikote> emaitza= new ArrayList<Bikote>();
237         SortedSet<Bikote> set = new TreeSet<Bikote>();
238         for(int i=0;i<al.size();i++){
239             String unekoStr=al.get(i);
240             Double unekoDou=this.aurrekoPageRank.get(unekoStr);
241             set.add(new Bikote(unekoStr,unekoDou));
242         }
243         Iterator<Bikote> it = set.iterator();
244         while (it.hasNext()) {
245             Bikote bik = it.next();
246             emaitza.add(bik);
247         }
248         return emaitza;
249     }
250 }
251
252 public void printOrdenean(ArrayList<Bikote> ordena, String s){
253     DecimalFormat df = new DecimalFormat("0",
254         ↳ DecimalFormatSymbols.getInstance(Locale.ENGLISH));
255     df.setMaximumFractionDigits(340);
256
257     System.out.println(s+"-(r)ekin erlazionatutako PageRank lista ordenatua:");
258
259     for (int i=0;i<ordena.size();i++){
260         System.out.println(i+1 + " " +ordena.get(i).aktoreaEdoPelikula+" PageRank
261         ↳ balioa: "+df.format(ordena.get(i).pageRank));
262     }
263 }
264 }
```

5.2.2 *Bikote.java*

```
1 package dea4;
2
3 public class Bikote implements Comparable<Bikote> {
4
5     String aktoreaEdoPelikula;
6     Double pageRank;
7
8     public Bikote(String unekoStr, Double unekoDou) {
9         this.aktoreaEdoPelikula=unekoStr;
10        this.pageRank=unekoDou;
11    }
12
13    public Double getPage(){
14        return this.pageRank;
15    }
16
17    @Override
18    public int compareTo(Bikote arg0) {
19        // TODO Auto-generated method stub
20        if((arg0.getPage() - this.pageRank)>0){return 1;}
21        else if((arg0.getPage() - this.pageRank)<0){return -1;}
22        else if((arg0.getPage() - this.pageRank)>0){return 0;}
23        return 1;
24    }
25
26 }
```

6. JUnitak

6.1 AURREKO EGINKIZUNEN JUNITAK

6.1.1 *AktoreTest.java*

```
1 package aktorePelikulaPackage;
2 import static org.junit.Assert.*;
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import org.junit.After;
6 import org.junit.Before;
7 import org.junit.Test;
8 import aktorePelikulaPackage.Aktore;
9
10 public class AktoreTest {
11
12     Aktore aktore1, aktor1;
13     ArrayPelikulak lista, lista1;
14     Pelikula peli1;
15     Pelikula peli2;
16
17     @Before
18     public void setUp() throws Exception {
19         aktore1 = new Aktore("Adeiarrias");
20         lista = new ArrayPelikulak();
21         peli1=new Pelikula("El Guason",45.00);
22         peli2=new Pelikula ("El Joker", 60.00);
23     }
24
25     @After
26     public void tearDown() throws Exception {
27         aktore1=null;
28         lista=null;
29         peli1=null;
30         peli2=null;
31     }
32
33     @Test
34     public void testGetIzenAbizena() throws FileNotFoundException, IOException {
35         assertEquals(aktore1.getIzenAbizena(), "Adeiarrias");
36         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
```

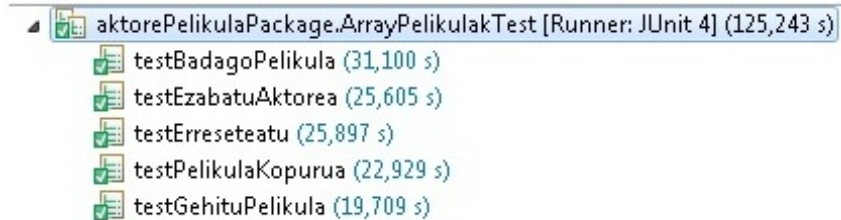
```
37     aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Baskin, Cezmi");
38     assertEquals(aktor1.getIzenAbizena(), "Baskin, Cezmi");
39 }
40
41 @Test
42 public void testGetListaPelikula() throws FileNotFoundException, IOException {
43     lista=aktore1.getListaPelikula();
44     assertNotNull(lista);
45     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
46     aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Baskin, Cezmi");
47     lista1 = aktor1.getListaPelikula();
48     assertNotNull(lista1);
49 }
50
51 @Test
52 public void testListaPelikulanGehitu() throws FileNotFoundException, IOException {
53     aktore1.listaPelikulanGehitu(peli1);
54     aktore1.listaPelikulanGehitu(peli2);
55     lista=aktore1.getListaPelikula();
56     assertTrue(lista.badagoPelikula(peli1.getIzenburua()));
57     assertTrue(lista.badagoPelikula(peli1.getIzenburua()));
58     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
59     Aktore aktor = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Baskin,
60     ↪ Cezmi");
61     lista = aktor.getListaPelikula();
62     assertEquals(lista.pelikulaKopurua(), 29);
63     lista.gehituPelikula(peli1);
64     assertEquals(lista.pelikulaKopurua(), 30);
65 }
```



6.1.2 *ArrayPelikulakTest.java*

```
1 package aktorePelikulaPackage;
2 import static org.junit.Assert.*;
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import org.junit.After;
6 import org.junit.Before;
7 import org.junit.Test;
8
9 public class ArrayPelikulakTest {
10
11     ArrayPelikulak lista, lista2;
12     Pelikula peli1, peli2, peli3, peli4;
13     Aktore aktor1;
14     ArrayAktoreak lista3;
15
16     @Before
17     public void setUp() throws Exception {
18         lista2 = new ArrayPelikulak();
19         peli1 = new Pelikula("300", 245.00);
20         peli2 = new Pelikula("Annabelle", 47.99);
21         peli3 = new Pelikula("Jurassic Park", 45.00);
22         lista2.gehituPelikula(peli2);
23         lista2.gehituPelikula(peli3);
24     }
25
26     @After
27     public void tearDown() throws Exception {
28
29     }
30
31     @Test
32     public void testBadagoPelikula() throws FileNotFoundException, IOException {
33         assertTrue(lista2.badagoPelikula("Annabelle"));
34         assertFalse(lista2.badagoPelikula("Eager to Die"));
35         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
36         aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Devon, Tony");
37         lista = aktor1.getListaPelikula();
38         assertTrue(lista.badagoPelikula("Eager to Die"));
39         assertFalse(lista.badagoPelikula("The Cold Shoulder"));
40     }
41
42     @Test
```

```
43 public void testGehituPelikula() throws FileNotFoundException, IOException {
44     assertEquals(lista2.pelikulaKopurua(), 2);
45     lista2.gehituPelikula(peli3);
46     assertEquals(lista2.pelikulaKopurua(), 3);
47     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
48     aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Devon, Tony");
49     lista = aktor1.getListaPelikula();
50     lista.erreseteatu();
51     assertEquals(lista.pelikulaKopurua(), 0);
52     lista.gehituPelikula(peli1);
53     assertEquals(lista.pelikulaKopurua(), 1);
54 }
55
56 @Test
57 public void testPelikulaKopurua() throws FileNotFoundException, IOException {
58     assertEquals(lista2.pelikulaKopurua(), 2);
59     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
60     aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Devon, Tony");
61     lista = aktor1.getListaPelikula();
62     assertEquals(lista.pelikulaKopurua(), 4);
63 }
64
65 @Test
66 public void testErreseteatu() throws FileNotFoundException, IOException {
67     assertEquals(lista2.pelikulaKopurua(), 2);
68     lista2.erreseteatu();
69     assertEquals(lista2.pelikulaKopurua(), 0);
70     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
71     aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Devon, Tony");
72     lista = aktor1.getListaPelikula();
73     lista.erreseteatu();
74     assertEquals(lista.pelikulaKopurua(), 0);
75 }
76
77 @Test
78 public void testEzabatuAktorea() throws FileNotFoundException, IOException {
79     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
80     aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Devon, Tony");
81     lista = aktor1.getListaPelikula();
82     peli4 = ListaPelikula.getNireListaPelikula().bilatuPelikula("The Wire");
83     lista3 = peli4.getListaAktoreak();
84     assertEquals(lista3.aktoreKopurua(), 702);
85     lista.ezabatuAktorea(aktor1);
86     assertEquals(lista3.aktoreKopurua(), 701);
87 }
88 }
```



6.1.3 ArrayAktoreakTest.java

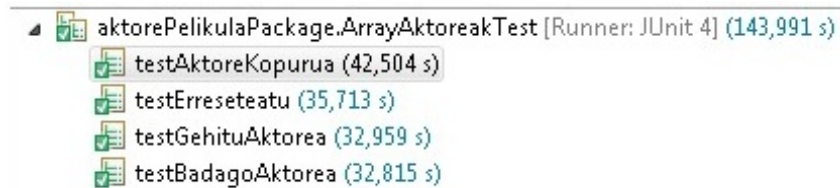
```
1 package aktorePelikulaPackage;
2 import static org.junit.Assert.*;
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import org.junit.After;
6 import org.junit.Before;
7 import org.junit.Test;
8
9 public class ArrayAktoreakTest {
10
11     ArrayAktoreak lista;
12     Aktore aktor1, aktor2, aktor3;
13     Pelikula peli1;
14
15     @Before
16     public void setUp() throws Exception {
17         lista = new ArrayAktoreak();
18         aktor1 = new Aktore("AdeiArias");
19         aktor2 = new Aktore("JonBarbero");
20         aktor3 = new Aktore("AnderPrieto");
21     }
22
23     @After
24     public void tearDown() throws Exception {
25         lista=null;
26         aktor1=null;
27         aktor2=null;
28         aktor3=null;
```

```
29     }
30
31
32
33
34     @Test
35     public void testBadagoAktorea() throws FileNotFoundException, IOException {
36         lista.erreseteatu();
37         lista.gehituAktorea(aktor1);
38         assertEquals(lista.badagoAktorea("AdeiArias"), true);
39         assertEquals(lista.badagoAktorea("JonBarbero"), false);
40         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
41         peli1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("The Ghost of the
42             ↪ Castle");
43         lista = peli1.getListaAktoreak();
44         assertTrue(lista.badagoAktorea("Foti, Leo"));
45         assertFalse(lista.badagoAktorea("Tejada, Beatriz"));
46     }
47
48     @Test
49     public void testGehituAktorea() throws FileNotFoundException, IOException {
50         lista.erreseteatu();
51         assertEquals(lista.aktoreKopurua(), 0);
52         lista.gehituAktorea(aktor2);
53         assertEquals(lista.aktoreKopurua(), 1);
54         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
55         peli1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("The Ghost of the
56             ↪ Castle");
57         lista = peli1.getListaAktoreak();
58         assertEquals(lista.aktoreKopurua(), 17);
59         lista.gehituAktorea(aktor1);
60         assertEquals(lista.aktoreKopurua(), 18);
61     }
62
63     @Test
64     public void testAktoreKopurua() throws FileNotFoundException, IOException {
65         lista.erreseteatu();
66         assertEquals(lista.aktoreKopurua(), 0);
67         lista.gehituAktorea(aktor2);
68         assertEquals(lista.aktoreKopurua(), 1);
69         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
70         peli1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("The Ghost of the
71             ↪ Castle");
72         lista = peli1.getListaAktoreak();
```

```

72     assertEquals(lista.aktoreKopurua(), 17);
73 }
74
75
76
77 @Test
78 public void testErreseteatu() throws FileNotFoundException, IOException {
79     lista.erreseteatu();
80     assertEquals(lista.aktoreKopurua(), 0);
81     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
82     peli1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("The Ghost of the
83         ↪ Castle");
84     lista = peli1.getListaAktoreak();
85     lista.erreseteatu();
86     assertEquals(lista.aktoreKopurua(), 0);
87 }

```



```

▲ aktorePelikulaPackage.ArrayAktoreakTest [Runner: JUnit 4] (143,991 s)
  ✔ testAktoreKopurua (42,504 s)
  ✔ testErreseteatu (35,713 s)
  ✔ testGehituAktorea (32,959 s)
  ✔ testBadagoAktorea (32,815 s)

```

6.1.4 *ListaAktoreakTest.java*

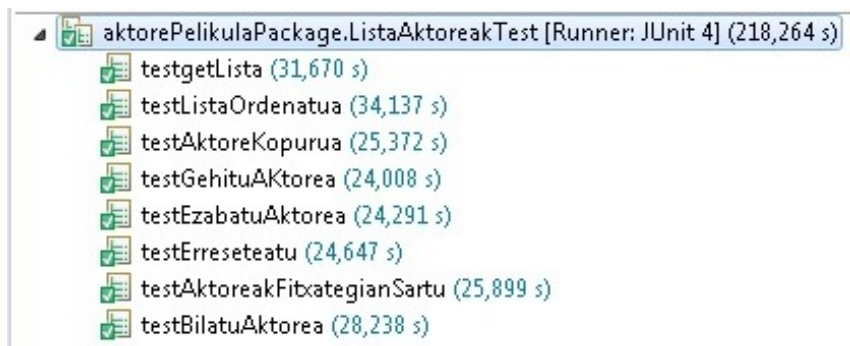
```

1 package aktorePelikulaPackage;
2 import static org.junit.Assert.*;
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import org.junit.After;
6 import org.junit.Before;
7 import org.junit.Test;
8 import aktorePelikulaPackage.Aktore;
9 import aktorePelikulaPackage.ListaAktoreak;
10
11 public class ListaAktoreakTest {
12

```

```
13 ListaAktoreak lista1 = ListaAktoreak.getNireListaAktoreak();
14 ListaPelikula lista2=ListaPelikula.getNireListaPelikula();
15 Aktore aktor1,aktor2;
16
17
18 @Before
19 public void setUp() throws Exception {
20     aktor1 = new Aktore("AdeiArias");
21     aktor2 = new Aktore("AnderPrieto");
22     lista2.fitxeroaErakutsi("./FilmsActors20162017.txt");
23 }
24
25 @After
26 public void tearDown() throws Exception {
27     aktor1=null;
28     aktor2=null;
29     lista1.erreseteatu();
30     lista2=null;
31 }
32
33 @Test
34 public void testEzabatuAktorea() throws FileNotFoundException, IOException {
35     assertEquals(lista1.aktoreKopurua(), 1283445);
36     lista1.ezabatuAktorea("AdeiArias");
37     assertEquals(lista1.aktoreKopurua(), 1283445);
38     lista1.ezabatuAktorea("Devon, Tony");
39     assertEquals(lista1.aktoreKopurua(), 1283444);
40 }
41
42 @Test
43 public void testGehituAktorea() throws FileNotFoundException, IOException {
44     assertEquals(lista1.aktoreKopurua(), 1283445);
45     lista1.gehituAktorea("AnderPrieto", aktor2);
46     assertEquals(lista1.aktoreKopurua(), 1283446);
47 }
48
49 @Test
50 public void testBilatuAktorea() {
51     lista1.gehituAktorea("AdeiArias", aktor1);
52     assertEquals(lista1.bilatuAktorea("AdeiArias"), aktor1);
53     assertEquals(lista1.bilatuAktorea("AnderPrieto"), null);
54     assertNotEquals(lista1.bilatuAktorea("Tarantino, Quentin"), aktor1);
55 }
56
57 @Test
58 public void testgetList() {
```

```
59     lista1.getList();
60     assertNotNull(lista1);
61 }
62
63
64 @Test
65 public void testAktoreKopurua() {
66     assertEquals(lista1.aktoreKopurua(), 1283445);
67 }
68
69 @Test
70 public void testErreseteatu() {
71     assertEquals(lista1.aktoreKopurua(), 1283445);
72     lista1.erreseteatu();
73     assertEquals((lista1.aktoreKopurua()), 0);
74 }
75
76 @Test
77 public void testListaOrdenatua() {
78     lista1.listaOrdenatua();
79     assertNotNull(lista1);
80 }
81
82 @Test
83 public void testAktoreakFitxategianSartu() {
84     lista1.aktoreakFitxategianSartu();
85 }
86 }
```



6.1.5 *ListaPelikulaTest.java*

```
1 package aktorePelikulaPackage;
2 import static org.junit.Assert.*;
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import org.junit.After;
6 import org.junit.Before;
7 import org.junit.Test;
8 import aktorePelikulaPackage.ListaAktoreak;
9 import aktorePelikulaPackage.ListaPelikula;
10 import aktorePelikulaPackage.Pelikula;
11
12 public class ListaPelikulaTest {
13
14     ListaPelikula lista1 = ListaPelikula.getNireListaPelikula();
15     Pelikula peli1, peli2, peli3, peli4;
16
17     @Before
18     public void setUp() throws Exception {
19         peli1 = new Pelikula("Batman", 345.00);
20         peli2 = new Pelikula("Joker", 355.00);
21         peli3 = new Pelikula("WonderWoman", 365.00);
22     }
23
24     @After
25     public void tearDown() throws Exception {
26     }
27
28     @Test
29     public void testGehituDirua() throws FileNotFoundException, IOException {
30         lista1.erreseteatu();
31         lista1.gehituPelikula(peli3);
32         lista1.gehituDirua("WonderWoman", 20.00);
33         assertEquals(peli3.getDirua(), 385.00, 2); //listako elementu bakarrari dirua
           ↳ gehitu
34         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
35         peli4 = ListaPelikula.getNireListaPelikula().bilatuPelikula("Eager to Die");
36         lista1.gehituDirua("Eager to Die", 20.00);
37         assertEquals(peli4.getDirua(), 65.00, 2); //listako edozein elementuri dirua
           ↳ gehitu
38     }
39
40     @Test
```



```

41 public void testPelikulaKopurua() throws FileNotFoundException, IOException {
42     lista1.erreseteatu();
43     assertEquals(lista1.pelikulaKopurua(), 0); //lista hutsaren pelikula kopurua
44     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
45     assertEquals(lista1.pelikulaKopurua(), 10412);
46     lista1.gehituPelikula(peli2);
47     assertEquals(lista1.pelikulaKopurua(), 10413); //lista ez hutsaren pelikula
         ↪ kopurua
48 }
49
50 @Test
51 public void testErreseteatu() throws FileNotFoundException, IOException {
52     lista1.erreseteatu();
53     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
54     assertEquals(lista1.pelikulaKopurua(), 10412);
55     lista1.erreseteatu();
56     assertEquals(lista1.pelikulaKopurua(), 0);
57 }
58
59 @Test
60 public void testGehituPelikula() throws FileNotFoundException, IOException {
61     lista1.erreseteatu();
62     assertEquals(ListaPelikula.getNireListaPelikula().pelikulaKopurua(), 0);
63     lista1.gehituPelikula(peli2);
64     assertEquals(ListaPelikula.getNireListaPelikula().pelikulaKopurua(),
        ↪ 1); //gehitu elementua lista hutsean
65     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
66     assertEquals(ListaPelikula.getNireListaPelikula().pelikulaKopurua(), 10413);
67     lista1.gehituPelikula(peli3);
68     assertEquals(ListaPelikula.getNireListaPelikula().pelikulaKopurua(),
        ↪ 10414); //gehitu elementua lista ez hutsean
69 }
70
71 @Test
72 public void testBilatuPelikula() throws FileNotFoundException, IOException {
73     lista1.erreseteatu();
74     assertEquals(ListaPelikula.getNireListaPelikula().bilatuPelikula("Eager to
        ↪ Die"), null); //lista hutsean bilatu
75     lista1.gehituPelikula(peli3);
76     assertEquals(ListaPelikula.getNireListaPelikula().bilatuPelikula("WonderWoman"),
        ↪ peli3); //lista elementu bakarra
77     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
78     lista1.gehituPelikula(peli1);
79     assertEquals(lista1.bilatuPelikula("Batman"), peli1); //elementua listan dago
80     assertNotEquals(lista1.bilatuPelikula("Joker"), peli1); //elementua ez dago
        ↪ listan

```

```

81     }
82
83     @Test
84     public void testFitxeroaErakutsi() throws FileNotFoundException, IOException {
85         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
86     }
87 }

```



6.1.6 PelikulaTest.java

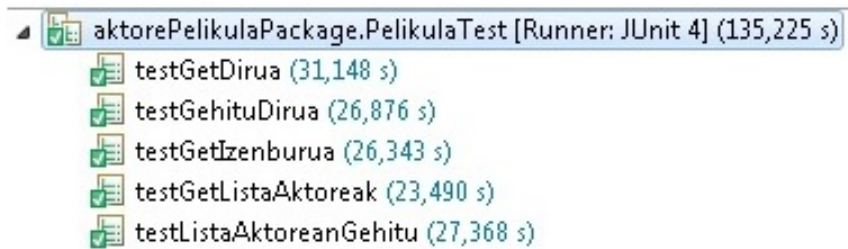
```

1  package aktorePelikulaPackage;
2  import static org.junit.Assert.*;
3  import java.io.FileNotFoundException;
4  import java.io.IOException;
5  import org.junit.After;
6  import org.junit.Before;
7  import org.junit.Test;
8  import aktorePelikulaPackage.ListaAktoreak;
9  import aktorePelikulaPackage.Pelikula;
10
11  public class PelikulaTest {
12
13      Pelikula pelikula1, pelikula2, pelikula3;
14      ArrayAktoreak lista, lista2;
15      Aktore aktor1, aktor2;
16
17      @Before
18      public void setUp() throws Exception {
19          pelikula3 = new Pelikula("La isla", 30.00);

```

```
20     pelikula2 = new Pelikula("Spiderman", 40.00);
21     lista2 = new ArrayAktoreak();
22     aktor1 = new Aktore("Arias, Adei");
23     aktor2 = new Aktore("Prieto, Ander");
24     lista2.gehituAktorea(aktor1);
25     lista2.gehituAktorea(aktor2);
26 }
27
28 @After
29 public void tearDown() throws Exception {
30 }
31
32 @Test
33 public void testGetIzenburua() throws FileNotFoundException, IOException {
34     assertEquals(pelikula2.getIzenburua(), "Spiderman");
35     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
36     pelikula1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("Mind Stroll");
37     assertEquals(pelikula1.getIzenburua(), "Mind Stroll");
38 }
39
40 @Test
41 public void testGetDirua() throws FileNotFoundException, IOException {
42     assertEquals(pelikula3.getDirua(), 30.00, 2);
43     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
44     pelikula1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("Mind Stroll");
45     assertEquals(pelikula1.getDirua(), 45.00, 2);
46 }
47
48 @Test
49 public void testGetListaAktoreak() throws FileNotFoundException, IOException {
50     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
51     pelikula1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("Mind Stroll");
52     lista = pelikula1.getListaAktoreak();
53     assertNotNull(lista);
54 }
55
56 @Test
57 public void testGehituDirua() throws FileNotFoundException, IOException {
58     pelikula3.gehituDirua(30.00);
59     assertEquals(pelikula3.getDirua(), 60.00, 2);
60     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
61     pelikula1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("La sala");
62     pelikula1.gehituDirua(10.00);
63     assertEquals(pelikula1.getDirua(), 55.00, 2);
64 }
65
```

```
66 @Test
67 public void testListaAktoreanGehitu() throws FileNotFoundException, IOException {
68     lista2.erreseteatu();
69     assertEquals(lista2.aktoreKopurua(), 0);
70     lista2.gehituAktorea(aktor1);
71     assertEquals(lista2.aktoreKopurua(), 1);
72     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
73     pelikula1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("Mind Stroll");
74     lista = pelikula1.getListAktoreak();
75     assertEquals(lista.aktoreKopurua(), 6);
76     lista.gehituAktorea(aktor1);
77     assertEquals(lista.aktoreKopurua(), 7);
78 }
79 }
```



```
aktorePelikulaPackage.PelikulaTest [Runner: JUnit 4] (135,225 s)
  testGetDirua (31,148 s)
  testGehituDirua (26,876 s)
  testGetIzenburua (26,343 s)
  testGetListaAktoreak (23,490 s)
  testListaAktoreanGehitu (27,368 s)
```

6.2 LAUGARREN EGINKIZUNEN UNITAK

6.2.1 *GraphHashTest.java*

```
1 package dea4;
2
3 import static org.junit.Assert.*;
4
5 import java.util.ArrayList;
6 import java.util.HashMap;
7
8 import org.junit.After;
9 import org.junit.Before;
10 import org.junit.Test;
11
12 public class GraphHashTest {
13     GraphHash grafoa;
14     ListaAktoreak lista1 = ListaAktoreak.getNireListaAktoreak();
15     ListaPelikula lista2=ListaPelikula.getNireListaPelikula();
16     HashMap<String,Double> aurrekoPageRank = new HashMap<String,Double>();
17     HashMap<String,Double> unekoPageRank = new HashMap<String,Double>();
18
19     @Before
20     public void setUp() throws Exception {
21         grafoa=new GraphHash();
22         lista2.fitxeroaErakutsi("./FilmsActors20162017.txt");
23     }
24
25     @After
26     public void tearDown() throws Exception {
27     }
28
29     @Test
30     public void testGrafoaSortu() {
31         grafoa.grafoaSortu(ListaAktoreak.getNireListaAktoreak());
32         assertNotNull(grafoa);
33     }
34
35     @Test
36     public void testPrint() {
37         grafoa.print();
38     }
39 }
```

```
40 @Test
41 public void testKonektatuta() {
42     grafoa.grafoaSortu(ListaAktoreak.getNireListaAktoreak());
43     assertTrue(grafoa.konektatuta("Devon, Tony", "Nutcher, Greg"));
44     assertFalse(grafoa.konektatuta("Devon, Tony", "Malas, Javi"));
45     assertFalse(grafoa.konektatuta("Pearson, Liam", "Bardem, Javier"));
46     assertFalse(grafoa.konektatuta("Neeson, Liem", "Pit, Brad"));
47     assertFalse(grafoa.konektatuta("Aho, Miina", "Pitt, Brad"));
48 }
49
50 @Test
51 public void testErlazionatuta() {
52     grafoa.grafoaSortu(ListaAktoreak.getNireListaAktoreak());
53     assertNotNull(grafoa.erlazionatuta("Devon, Tony", "O'Toole, Peter (I)"));
54     assertEquals(grafoa.erlazionatuta("Devon, Tony", "Maalas, Javii"), null);
55     assertEquals(grafoa.erlazionatuta("Peaarson, Liiam", "Bardem, Javier"), null);
56     assertEquals(grafoa.erlazionatuta("Neeson, Liem", "Pit, Brad"), null);
57     assertEquals(grafoa.erlazionatuta("Aho, Miina", "Pitt, Brad"), null);
58 }
59
60 @Test
61 public void testHasieratuPageRank() {
62     grafoa.hasieratuPageRank();
63 }
64
65 @Test
66 public void testPageRank() {
67     grafoa.pageRank();
68 }
69
70 @Test
71 public void testPrintPageRank() {
72     grafoa.printPageRank(unekoPageRank);
73 }
74
75 @Test
76 public void testBilatzailea() {
77     assertNotNull(grafoa.bilatzailea("McGregor, Ewan"));
78     assertNull(grafoa.bilatzailea("Peppe, Bootiijoo"));
79 }
80
81 @Test
82 public void testPrintOrdenean() {
83     ArrayList<Bikote> ordena = grafoa.bilatzailea("McGregor, Ewan");
84     grafoa.printOrdenean(ordena, "McGregor, Ewan");
85     ArrayList<Bikote> bilatu = grafoa.bilatzailea("Juanra, Giimeeneez");
```

```
86     if (bilatu!=null){
87         grafoa.printOrdenean(ordena,"Juanra, Giimeeenenez");
88     }
89     else{
90         System.out.println("Ez dago grafoan");
91     }
92 }
93
94 }
```

PRAKTIKAREN EXEKUZIOAREN ADIBIDEAREN HASIERA:

FITXATEGIA KARGATZEN ARI DA
8.43 segundu behar izan ditu

GRAFOA SORTZEN ARI DA
0.547 segundu behar izan ditu

PAGERANK GRAFOA SORTZEN ARI DA
175.438 segundu behar izan ditu; 59 iterazio behar izan ditu

ELEMENTUAREN LISTA SORTZEN ARI DA
0.01 segundu behar izan ditu

PRAKTIKAREN EXEKUZIOAREN ADIBIDEAREN AMAIERA.

7. Ondorioak

Azkenengo praktika hau aurrekoak baino gehiago kostatu zaigu, arazoak aurkitzeko zailegia baitzen. Hala ere, taldekideei esker arazo horiek zuzentzeko gai izan gara.

Lauhilabete honetan datu egitura ezberdinei buruzko aplikazio praktikoak ikusi ditugu; izan ere, duela pare bat hilabetetik hona asko hobetu dugu gure maila.

Orokorrean hitz eginez, frogatu dugu nolakoa den goi mailako programatzailea izatea, eta gure ustez lan bikaina egin dugu.

Erreferentziak

- [1] Gojenola, Koldo. Datu-Egiturak eta Algoritmoak: proiektua – Aktoreak eta pelikulak kudeatu – 4. eginkizuna (*PageRank*). egela.ehu.eus, 2019. URL https://egela.ehu.eus/pluginfile.php/2282887/mod_resource/content/9/Praktika%202019-2020%20ikasturtea-Fase4-euskaraz.pdf.
- [2] Ezezaguna. PageRank - Wikipedia, entziklopedia askea. eu.wikipedia.org, 2018. URL <https://eu.wikipedia.org/wiki/PageRank>.
- [3] Oracle. HashMap (Java Platform SE 7). docs.oracle.com, 2018. URL <https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html>.
- [4] Oracle. HashSet (Java Platform SE 7). docs.oracle.com, 2018. URL <https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html>.
- [5] Oracle. SortedSet (Java Platform SE 7). docs.oracle.com, 2018. URL <https://docs.oracle.com/javase/7/docs/api/java/util/SortedSet.html>.