

Datu-egiturak eta algoritmoak

Datu-egiturak eta algoritmoak

- Irakasleak: Eneko Iradier eta Koldo Gojenola
Bulegoak: 3I31 (Eneko), 3I32 (Koldo), eskolako 3.
solairuan
- Tutoratzak: ikusi UPV/EHUko webean
- Klaseak (teoria):
 - Astearte 15:00 - 17:00
 - Ostirala 15:00 - 16:00
- Laborategiak
 - Ostirala 16:00 - 18:00 (azpitralde 1 eta 2 txandakatuta)

Zertarako datu-egiturak?

**Computer Science Curriculum 2008:
Association for Computing Machinery / IEEE Computer Society**

[http://www.acm.org/education/curricula/
ComputerScience2008.pdf](http://www.acm.org/education/curricula/ComputerScience2008.pdf)

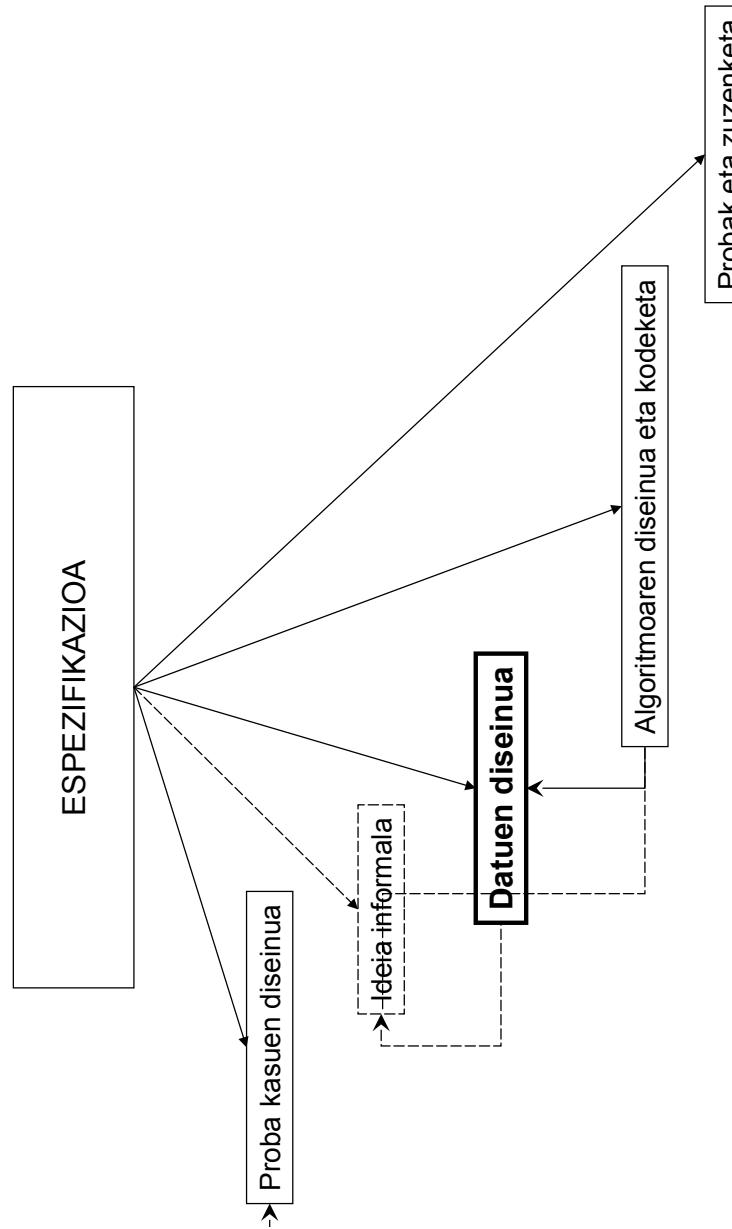
*The thing that we can't afford to do [...] is teach candidates how to think critically, to be effective problem solvers, and to have **basic mastery of programming languages, data structures, algorithms**, concurrency, networking, computer architecture, and discrete math / probability / statistics. **I can't begin to emphasize the importance of algorithms and data structures to the work we do here [...]. With multi-terabyte disks, bigger broadband pipes, etc. on the way, the big data problems that demand these skills [...] are quickly going to be in need in a huge number of programming contexts.***

Programen diseinua eta datuen diseinua

- Programak egitean, algoritmoak eta objektuen errepresentazioa estu lotuta daude
 - Problema bat ebazteko, datuen errepresentazio egokia aukeratzea ezinbesteko da
 - Diseinatzen diren algoritmoak datu-egituren araberaokoak dira
- Biak landu behar ditu programatzailreak

Programen garapenerako urratsak

1. Espezifikazioa (zehaztapena): parametrizazioa
2. Proba-kasuen diseinua
3. *Ebazpenerako ideia informala*
4. Datu-egituren diseinua
5. Programaren diseinua eta kodeketa
6. Probatzea eta zuzentzea



Programen garapenerako urratsak (III)

- **Espesifikazioa:** egin behar dena zehatz-mehatz adieraztea
 - Sintaxia:
 - Eragiketaren deskribapena: klase bateko metodoa
 - Parametroak
 - Bestelako xehetasunak
 - Semantika:
 - Aurrebaldintza: Datuek bete behar dituzten murriztapenak, programak bere lana ondo egin dezan
 - Postbaldintza: Emaitzek beteko dituzten propietateak

09/08/22

Datu-Egiturak eta Algoritmoak

7

Programen garapenerako urratsak (IV)

- Proba-kasuen diseinua
 - Gerta litezkeen kasuak eta eman beharko lituzketen emaitzak adieraztea
- Ebazpen-ideia informala
 - Ebazpenerako ideia bat pentsatu eta informalki adieraztea, gehiegizko xehetasunik gabe. Batzuetaan grafikoki adieraztea komeni da

09/08/22

Datu-Egiturak eta Algoritmoak

8

Programen garapenerako urratsak (V)

- **Datuene diseinua**
 - Datuak manejatzeko egitura egokiak adieraztea
 - Honakoan ere, adierazpide grafikoa egokia izaten da
 - Datu-egiturak mailaka ezarri behar dira. Horregatik, datu-egituren aukeraketa eta algoritmoen diseinua paraleloan egin ohi dira
 - Bereizi, kanpo-fitxategietako informazioa eta barneko datu-egiturak

09/08/22

Datu-Egiturak eta Algoritmoak

9

Programen garapenerako urratsak (VI)

- **Programaren diseinua eta kodeketa**
 - Beheranzko diseinua. Pausoka-pausoka, gero eta agindu simpleagoak jarriaz. Kodeketa, azken batean, diseinuaren azkeneko pausoa baizik ez da
- **Probak eta zuzenketa**
 - Proba-kasuetan oinarrituta, probatu eta zuzendu
 - Erroreen prebentzia eta detekzia
 - Egitura-probak eta proba funtzionalak
 - Integrazio-probak
 - Eraginkortasunari buruzko hausnarketa egin, aplikazioa hobetzeko

09/08/22

Datu-Egiturak eta Algoritmoak

10

Irakasgai gainditzerakoan ikasleak gai izan behar du...

- Diseinatutako **algoritmoen efizienzia** analizatzeko.
- **Datu-egitura nagusiak** (**listak**, **pilak**, **ilarak**, **zuhaitzak**, **grafoak eta hash taulak**) implementatu eta problema konputazionalen ebazpenean erabiltzeko.
- **Diseinu errekuentsiboa** erabiltzeko.
- **Bilaketa**, **ordenazio** eta datu-egituren korritze-algoritmo nagusiak analizatu eta erabiltzeko.
- Datu-egitura egokiak diseinatu eta implementatzeko, problemen ebazen eficiente bat lortzeko.
- Informatika Ingeniaritzako lanbideko ezagutzak eta trebetasunak komunikatzeko eta jakinarazteko gaitasuna izatea.

11

Metodologia

- Gaiak klasean azalduko ditugu, irakurtzeko material osagarria eskainiz.
- Teknika desberdinak menperatzeko ariketak egingo dira, betiere ikasleen parte-hartzea eta motibazioa bilatuz.
- Ikasturtean zehar, programazioko praktikak egingo dira. Proiektu baten betebeharak betetzeko aplikazio baten sorkuntza egin beharko da.
- Banakako nahiz taldeko lana bultzatuko da.

12

Gai-zerrrenda

1. gaiak: Algoritmoen analisia. Kostu-funtzioa.
2. gaiak: DMAak: listak, pilak eta ilarak.
 - Matrize-egiturak eta egitura estekatuak.
3. gaiak: Diseinu errekuertsihoa.
4. gaiak: Zuhaitzak. Bilaketako zuhaitz bitarrak. B,
B+ eta B* zuhaitzak.
5. gaiak: Grafoak. Grafoak korritzeko algoritmoak.
6. gaiak: Hash taulak. Hash funtzioa. Talken
ebazpena.
7. gaiak.- Problemen ebazpideen analisia, diseinua
eta implementazioa. Datu-egituren analisia eta
aukeraketa.

Aurkezpena

13

Bibliografia

- “Java Software Structures: Designing and Using Data Structures” (3rd edition)
John Lewis, Joseph Chase, Addison-Wesley (2010)
- Gaztelaniazko bertsioa badago:
“Estructuras de datos con Java: Diseño de estructuras
y algoritmos” (2a edición)
John Lewis y Joseph Chase, Addison-Wesley (2006)
- *Algorithms, 4th Edition,*
Robert Sedgewick and Kevin Wayne
<http://algs4.cs.princeton.edu/home/>

Aurkezpena

14

Tresnak

- Garapenerako (programatzeko) ingurune integratua:
 - <http://www.eclipse.org/downloads/>
 - IntelliJ Idea: <https://www.jetbrains.com/idea/>
- Ikasle-irakasle komunikazioako web-plataforma: <https://egela.ehu.eus>

Tresnak

- Dokumentazioak idazteko (latex):
<https://www.overleaf.com/>
- Kodea garatzeko eta partekatzeko:
<https://github.com/>
- Notebooks:
<https://jupyter.org/>
<https://www.anaconda.com/>

A Jupyter kernel for executing Java code:
<https://github.com/SpencerPark/IJava>

Helbide interesgarrak

- Java™ 2 Platform, Standard Edition, v 1.4.2 API Specification:
<http://download.oracle.com/javase/1.4.2/docs/api/>
- Dictionary of Algorithms and Data Structures:
<http://xlinux.nist.gov/dads/>

Aurkezpena

17

Helbide interesgarrak

Programazio-ariketak:<https://uva.onlinejudge.org/><https://codingcompetitions.withgoogle.com/hashcode/>**Free online courses:**<https://www.coursera.org/><https://www.coursera.org/courses?query=data%20structures%20and%20algorithms>

Advanced Data Structures in Java (University of California, San Diego)
Algorithms, Part I (Princeton), Algorithms, Part II

Algorithms: Design and Analysis, Part 1 (Stanford)

Aurkezpena

18

Tips to be a competitive programmer

DEA

Competitive Programming 3 [Steven & Felix Halim 2013]:

- Type code faster!
- Quickly identify problem types
- Do algorithm analysis
- Master programming languages
- Master the art of testing code
- Practice and more practice
- Team work

Sarrera

19.

DEA

Ebaluazioa

Bi ebaluazio-modu daude: ebaluazio jarraia edo azken ebaluazioa.

a) **Ebaluazio jarraia** egiteko, ikasleak irakasgaiari dagozkiion dedikazioa eta asistentzia ziurtatu beharko ditu. Hala, oinarrizko programazioko jakintza minimo bat erakutsi beharko du. Ikarsturte hasieran ebaluazio-modu hau aukeratu ala ez erabaki beharko dute ikasleek eta ikasturtearen % 70a egina dugunean, ebaluazio-modu jarraia onetsi edo bertan behera utzi ahal izango du ikasleak, irakaslearen gomendioei jarraiki, baina erabaki hau ezingo da aldatu une horretatik aurrera. Ebaluazio jarraian kontuan hartuko diren lanak hauek izango dira:

Programazioko praktika. Lan honen nota ikaslearen nota osoaren % 40a izango da, eta ikasturtean zehar iegindako lanek osatuko dute.

Banakako ebaluazioak. Klasean ikusitako galei buruzko kontrolak egingo dira, eta kontrol hauen emaitzen batzazbestekoia izango da nota osoaren gainontzeko % 60a.

- Lehen ebaluazioa klaseko 7. astean (15%)
- Bigarren ebaluazioa klaseko 11. astean (15%)
- Azken ebaluazioa azterketa garaien (30%)

(Irakasleak puntu baterainoko igoera eman ahal izango die, dedikazio, jarrera eta parte-hartze bikainak dituzten ikasleel)

b) **Azken ebaluazioa** aukeratzenten idatzia egingo zaie.

Aurkezpena

20

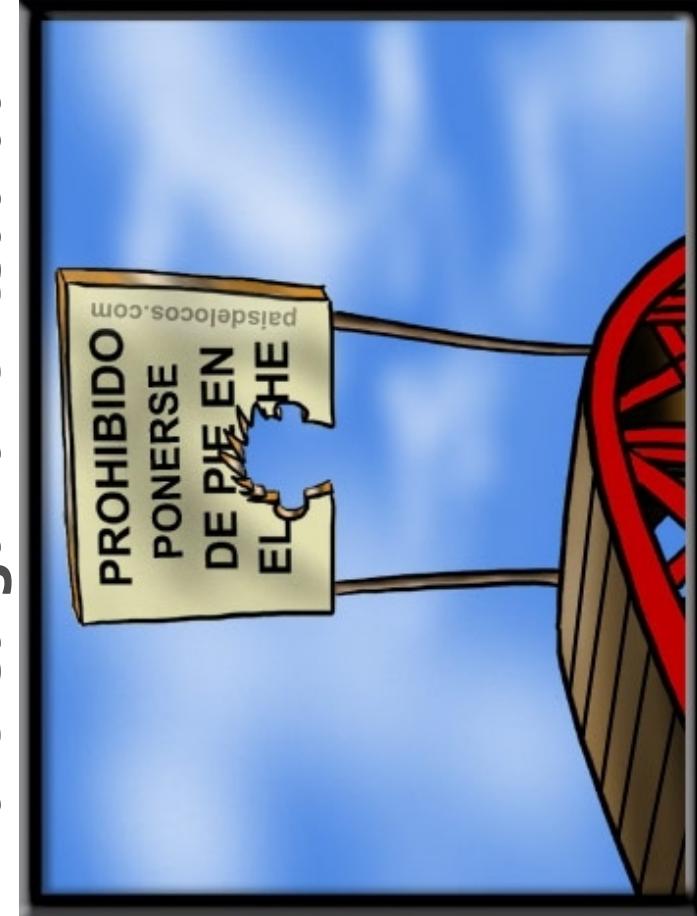
Ohar garrantzitsuak!

- Asistentzia oso garrantzitsua da
- Zalantzak lehenbailehen argitzea oso garrantzitsua da
- Ariketa ASKO egitea oso garrantzitsua da

Aurkezpena

21

Ohar garrantzitsuak kontuan hartzea garrantzitsua da!



Aurkezpena

22



Algoritmoen analisia

**Algoritmoen konplexutasunaren neurketa:
Bilatze- eta ordenatze-algoritmoen analisia**



Aurkibidea

- Zergatik da beharrezko?
- Nola neurtu exekuzio-denbora?

Zeren mende dago?

Nola kalkulatu?

Experimentalki

Matematikoki

- Algoritmoen analisia

Oinarrizko eragiketak

Notazio asintotikoa

- Analisi asintotikoaren murriztapenak

- Bilatze- eta ordenazio-algoritmoak eta beren konplexutasunaren neurketa

Txanpon faltsuak gehiago pisatzen du

-Sea B la bolsa de las monedas

algoritmo MonedaFalsa(B)

 m := Tomamonedade(B)

 PonEnPlatillo1(m)

 n := TomaMonedaDe(B)

 PonEnPlatillo2(n)

mientras PesaIgual1y2? **iterar**

 VaciaPlatillo2

 n := TomaMonedadde(B)

 PonEnPlatillo2(n)

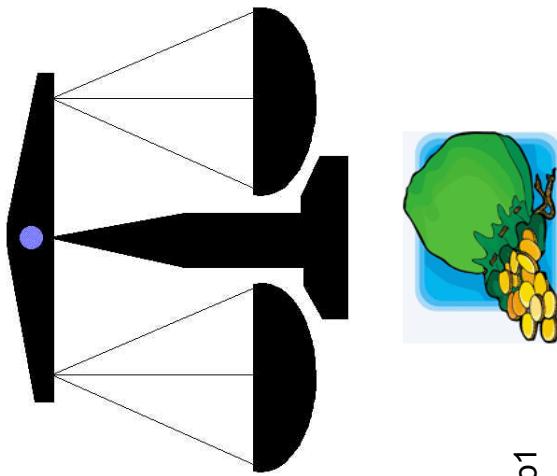
fin mientras

si PesaMas1? **entonces**

el resultado es MonedaDePlatillo1

si no

el resultado es MonedaDePlatillo2



3

Analisirako oinarrizko kontzeptuak

- **Sarreraren tamaina**
- **Oinarrizko eragiketa:**
 - Denbora konstantean exekutatzten den eragiketa, sarreraren tamainarekiko independentea
- **Denboraren kostu-funtzioa**
 - $f(n)$: n tamainako sarrera batekin egiten den oinarrizko eragiketen kopuruua
- **Eragiketa bereizgarria:**
 - Kostu-funtzioaren hazkunde-tasa ematen duen oinarrizko eragiketa (adibidez, PesaIgual1y2?)

4

Analisia

- Kostu-funtzioa zein den jakin behar dugu
 - $f(n) = \text{PesaIgual1y2?}$ Zenbat aldiz egiten den, n txanponeko poltsa batentzat
- Analisia kasu onenean $f_{\text{onena}}(n) = 1$
- Analisia kasu **txarrenean** $f_{\text{txarrena}}(n) = n-1$

5

Txanpon faltsua (revisited)

```
Moneda_Falsa_DYV(B)
m ← cardinal(B)
iterar
conjunto_1 ← Toma_monedas([ $\frac{m}{2}$ ] , B)
Pon_en_platillo_1(conjunto_1)
conjunto_2 ← Toma_monedas([ $\frac{m}{2}$ ] , B)
Pon_en_platillo_2(conjunto_2)
si ocurre que
Pesa igual 1.y.2:
    el resultado es Moneda_apartada
Pesa_mas_1:
    si m = 1 entonces
        el resultado es Moneda_de_platillo_1
    m ← [ $\frac{m}{2}$ ]
    B ← conjunto_1
    Vacia_platillo_2
    Pesa_mas_2:
        si m = 1 entonces
            el resultado es Moneda_de_platillo_2
        m ← [ $\frac{m}{2}$ ]
        B ← conjunto_2
        Vacia_platillo_1
fin iterar
```

6

Konparaketa

Txanpon-kopurua Pisaketa-kopurua

n	$n-1$	$\lfloor \log_2 n \rfloor$
32	31	5
128	127	7
1024	1023	10
1048 576	1047575	20
1073741824	1073741823	30
$1,1 \times 10^{12}$	$1,1 \times 10^{12} - 1$	40

7



Konplexutasuna

Zergatik da beharrezkoak konplexutasuna aztertzea?

Algoritmo bat garatu eta zuzena den frogatu ondoren.
bere **konplexutasun konputazionala** zehaztu behar
da (exekuziorako behar dituen baliabideak)

Nola neurtu?

- Exekuzio denbora
- Hartzen duen espazioa
- Memorian
- Diskoan



Konplexutasuna

Nola neurtu exekuzio-denbora?

Zer neurtzen da? Exekuzio-denbora
Zeren mende dago?

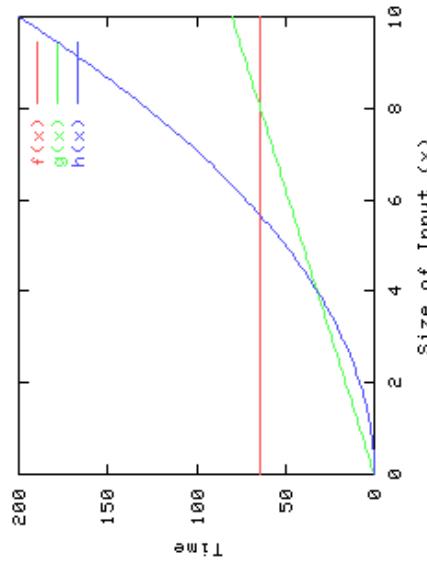
Sarrerako datuen tamainaren arabera
Beste hainbat faktore (Hw & Sw)
Ordenadorearen abiadura
Konpiladoreen kalitatea
Programaren kalitatea

Nola neurtu?
Experimentalki
Matematikoki estimatu

9

Algoritmoen konplexutasunaren analisia

- Nola neurten dugu algoritmoen efizientzia?
 - **Kostu-funtzioekin (denborakoak edo espaziokoak)**



Sarrera

10



Konplexutasuna

Nola neurri? Esperimentalki

Exekuzio denbora neurruz, sarrerako datuen tamainaren arabera

- Ezin ditugu sarrera posible guztiak frogatu
- Algoritmoa implementatzea beharrezkoa da
- Softwarearen eta hardwarearen mende dago

Beste neurri bat behar dugu:

- Abstraktuagoa
- Lortzen errazagoa

11

Kostu-funtzioaren ordena

- Kostu-funtzioak ordenatan multzokatzen dira
- Kostu-funtzioaren ordena:
 - $T(n)$ emanda, bere ordena da **hazkunde-eredu antzekoa duten funtzioen multzoa**

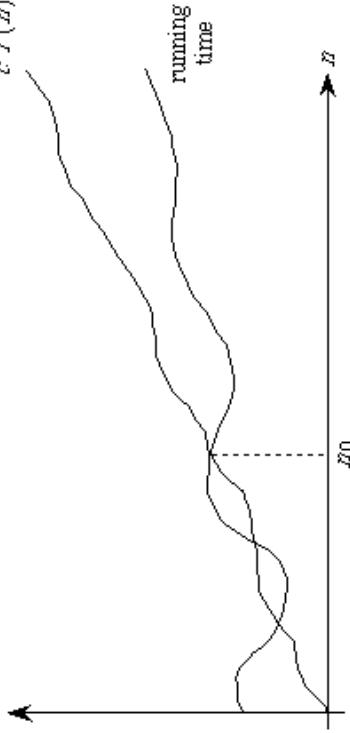
Notazio asintotikoa

Notación O (O grande de f)

$$O(f(n)) = \{g(n) \mid \exists c > 0 \exists n_0 \forall n \geq n_0 g(n) \leq cf(n)\}$$

$$g(n) \in O(1) \equiv \exists c > 0 \exists n_0 \forall n \geq n_0 g(n) \leq c$$

$$c f(x)$$



13

Ordenak

- Kostu-funtzioak taldekatzeko notazioak:

- $\Omega(T(n)) \equiv T(n)$ edo okerragoak
 - $\Theta(T(n)) \equiv T(n)$ bezalakoak
 - **$O(T(n)) \equiv T(n)$** edo hobeak
-

Portaera asintotikoa. Murriztapenak

- Algoritmoen analisiak, baldintza “gogorrak” aztertzea eskatzen du:
 - Sarrerako datuen tamaina infiniturantz
 - Portaera asintotikoa
- Baino, kontuan hartu behar da:
 - Sarrerako datuak gutxi direnean, ez da egokia
 - Analisi asintotikoa goi-estimazio bat izan ohi da
 - Exekuzio-denbora: programaren batezbesteko denbora, kasurik okerrenero exekuzio-denbora baino askoz txikiagoa izan daitete
 - Batzuetan kasu okerrena ez da esanguratsua.

08/09/2022

Datu-Egiturak eta Algoritmoeak

15

Kostu-funtzioetatik ordenatara Adibideak

- $T(n) = 2n + n^2 \rightarrow O(n^2)$
- $T(n) = 52 + 2n \rightarrow O(n)$
- $T(n) = 2n^2/5 + 6n + 3\pi \log_2 n + 2 \rightarrow O(n^2)$

08/09/2022

Datu-Egiturak eta Algoritmoeak

16

Ordene ezagunak

- $O(1)$ konstantea
 - $O(\log n)$ logaritmikoa
 - $O(n)$ lineala
 - $O(n \log n)$ quasilineala
 - $O(n^2)$ koadratikoa
 - $O(n^3)$ kubikoa
 - $O(n^k)$ polinomikoa
 - $O(k^n)$ exponentziala
- +
- non k konstantea den eta n sarreraren tamaina

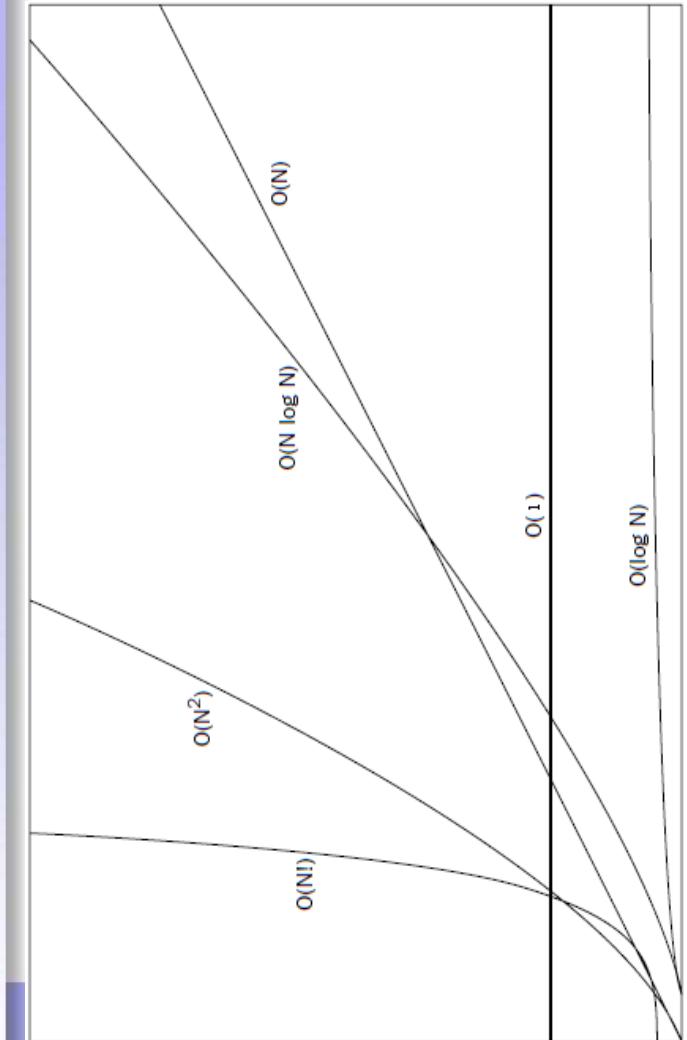
08/09/2022

Datu-Egiturak eta Algoritmoak

17



O Notazioa



18

Invariantza-legreak

- Batuketa

$$S(n) \in T(n) \Rightarrow K > 0 \Rightarrow$$

$$S(n) + T(n) \in O(T(n))$$

$$K * T(n) \in O(T(n))$$

08/09/2022

Datu-Egiturak eta Algoritmoak

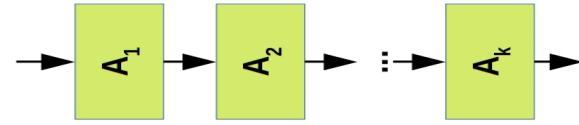
19

Sekuentziazioa

- Osagai guztien kostua kalkulatu
- **Batu** (denak exekutatu behar dira)

$$T_p(n) = T_{A_1}(n) + T_{A_2}(n) + \dots + T_{A_k}(n)$$

- Ordenari begira, batuketaren invariantza-legea aplikatu



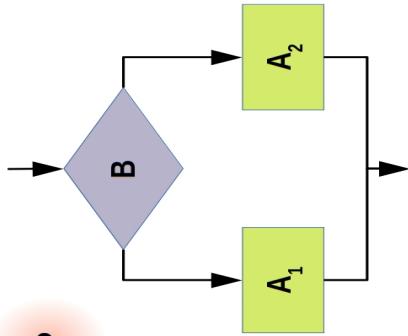
08/09/2022

Datu-Egiturak eta Algoritmoak

20

Adarkatzea (baldintzazko aginduaak)

- Alternatiben kostua kalkulatu eta haien **maximoa** hartu (kasurik txarrena)
- Baldintzaren kostua gehitu



$$T_p(n) = T_B(n) + \max(T_{A_1}(n), T_{A_2}(n))$$

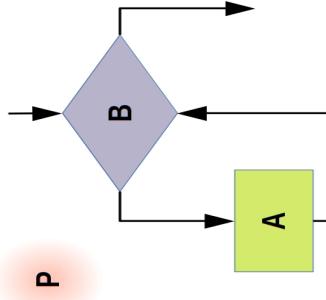
08/09/2022

Datu-Egiturak eta Algoritmoak

21

Iterazioa

- Demagun **k aldi** burutuko dela.
Aldi bakoitzean bai **B** bai **A** exekutatuko dira.
- Bi aukera:
 - Homogeneoa (iterazio guztietan kostu berdina)
 - Heterogeneoa (kostua aldatzen da iterazioen artean)



08/09/2022

Datu-Egiturak eta Algoritmoak

22

$$T_p(n) = \sum_{i=1}^k (T_{B_i}(n) + T_{A_i}(n))$$

$$T_p(n) = k^*(T_B(n) + T_A(n))$$

22

Programei egindako deiak

- Azpirutinak maiz erabiltzen ditugu
- Erraza da oharkabeen pasatzea
- Metodo-deietan, metodoen exekuzio-kostua **aparte** aztertu behar da

08/09/2022

Datu-Egiturak eta Algoritmoak

23

¿Garrantzitsua al da algoritmoen efizientzia?

Fibonacciren seriea: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

```
function Fib (N: Natural) return Natural is
begin
  if N<=1 then return N;
  else return Fib(N-1) + Fib(N-2);
  end if;
end
```

Sarrera

24

Zenbat batuketa egiten ditu Fib(N) deia?

- batuketak(N)= 1 + batuketak(N-1) + batuketak(N-2)
 $\forall N > 1$
 - batuketak(N-1)= 1 + batuketak(N-2) + batuketak(N-3)
 - batuketak(N) > 2×batuketak(N-2)
 - $2 \times \text{batuketak}(N-2) > 2 \times 2 \times \text{batuketak}(N-4) > \dots > 2^k \times \text{batuketak}(N-2^k)$
 - $\text{batuketak}(N) > 2^{N/2} = (\sqrt{2})^N$
- ```
function Fib (N: Natural) return Natural is
begin
 if N<=1 then return N;
 else return Fib(N-1) + Fib(N-2);
end if;
end;
```
- 25

Sarrera

## Denboren taula

| N   | $\sim 2^{n/2}$       | $2^{n/2} \times 10^{-9}$ segundu |
|-----|----------------------|----------------------------------|
| 50  | 33.554.432           | 33 mseg                          |
| 60  | $1,1 \times 10^9$    | 1 seg                            |
| 70  | $3,4 \times 10^{10}$ | 34 seg                           |
| 80  | $1,1 \times 10^{12}$ | 18 min                           |
| 90  | $3,5 \times 10^{13}$ | 9 ordú                           |
| 100 | $1,1 \times 10^{15}$ |                                  |
| 110 | $3,6 \times 10^{16}$ |                                  |
| 120 | $1,2 \times 10^{18}$ |                                  |
| 130 | $3,6 \times 10^{19}$ |                                  |

Sarrera

26

## Algoritmoen konplexutasuna



Demagun array baten lehenengo eta bigarren elementuak berdinak diren kalkulatzeko algoritmo bat dugula.

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j |
|---|---|---|---|---|---|---|---|---|---|

\* Zenbat komparaketa behar ditugu arrayak 10 elementu baditu?

\* Zenbat komparaketa behar ditugu arrayak 100 elementu baditu?



## Algoritmoen konplexutasuna

Algoritmoa, arrayaren neurriarekiko independentea da

Algoritmoaren konplexutasuna  $O(1)$  da

Egin behar den komparaketa kopurua konstantea da  
(ez dugu komparaketa gehiago egin behar, arrayan elementu gehiago izateagatik)



## Algoritmoen konplexutasuna

Array baten lehenengo elementua bigarrenaren edo hirugarrenaren berdina den kalkulatzeko algoritmoa:

|   |   |   |   |   |   |   |   |   |      |
|---|---|---|---|---|---|---|---|---|------|
| a | b | c | d | e | f | g | h | i | j    |
| ? | ? | ? |   |   |   |   |   |   | O(1) |

Egin behar den konparaketa kopurua konstantea da  
(ez dugu konparaketa gehiago egin behar, arrayan elementu gehiago izateagatik)



## Algoritmoen konplexutasuna

Array baten lehenengo elementua arrayan errepetituta dagoen ala ez aztertzen duen algoritmo bat.

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j |
| ↑ |   |   |   |   |   |   |   |   |   |

\* Kasu txarrenean zenbat konparaketa egin beharko ditugu?  
  
n – 1  
(n arrayaren elementu kopurua izanik)

n oso handia denean...  
  
→



## Konplexutasuna

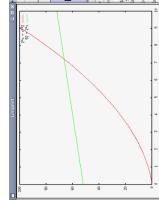
Nola neurtu? Estimazio matematikoa

Algoritmo bakoitzarentzat, bere **sarrerako datuen mende dagoen funtziobat** bilatzen da:  $f(n)$

Exekuzio denbora gisa, **kasu okerrena** hartuko da



Sarrerako datuen neurriaren arabera, **algoritmoeen hazkunde abiadura** kalkulatzea da helburua



31



## Konplexutasuna

Adibidea

Hurrengo metodoa garatu nahi bada:

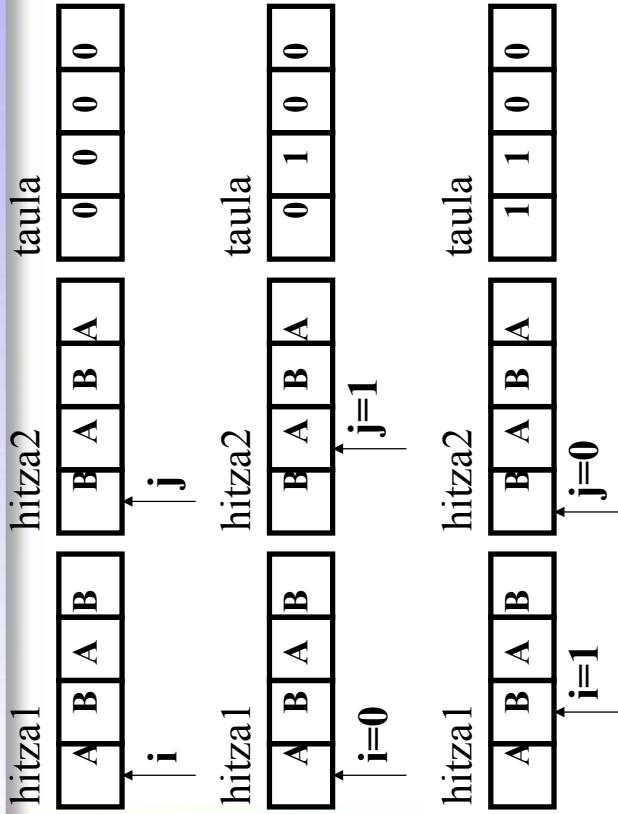
```
public boolean isAnagramma(Hitza h1, Hitza h2)
```

```
//aurre: h1 eta h2 letra xehez osatutako hitzak dira
//post: h1 eta h2 letra berdinez osatutako hitzak
// badira; false, bestela.
// Adibidea: h1="donostia" eta h2="itsaondo"
// anagramak dira
```

32



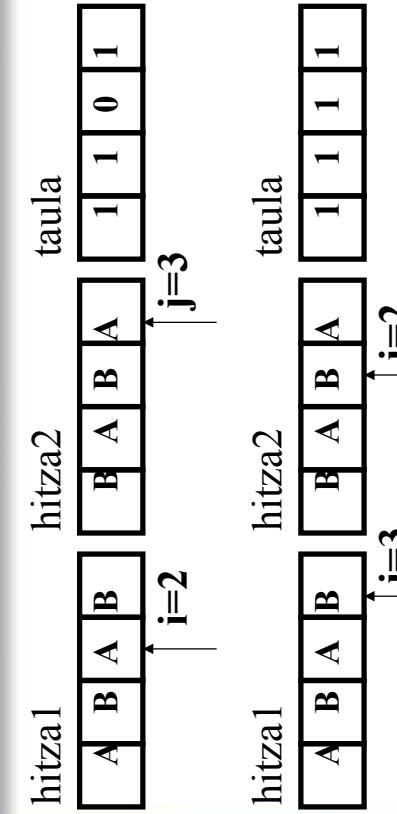
## **Adibidea : Anagrama? (I)**



33



## Adibidea: Anagramma? (1)



Egin heharreko eragiketak:

- 4 hasierako marka
  - hitzal -en letra guztiak hitza2-n bilatu:  
4\*4 konparaketa, kasurik okerrean  
- ikusi ea 4 letrak markatuak izan diren:  
kasurik okerrean, 4 konparaketa

34



## Anagramma()

Java programa

```
public static boolean isAnagramma(Hitza h1, Hitza h2) {

 int[] taula=new int[4]; int p,j;
 //Initialize visited positions
 for (int i=0;i<4;i++)
 taula[i]=0;
 for (int i=0;i<4;i++) {
 j=0;
 while ((j<4) && !(hitza1[i]==hitza2[j] && taula[j]==0))
 j++;
 if ((j<4) && hitza1[i]==hitza2[j] && taula[j]==0)
 taula[j]=1;
 }
 p=0;
 while ((p<4) && (taula[p]==1))
 p++;
 if (p==4) return true;
 else return false;
}
```

35

## AnagrammaDa 1: konplexutasunaren analisia

$$\begin{aligned} T(n) &= T_{\text{has}} + T_{\text{for1}} + T_{\text{for2}} + T_{\text{asig}} + T_{\text{wh1}} + T_{\text{ret}} \\ &= k_1 + n^*(T_{\text{bd}} + T_{\text{asig}}) + n^*(T_{\text{bd}} + T_{\text{asig}} + T_{\text{wh2}} + T_{\text{if}}) + k_2 + T_{\text{wh1}} + k_3 \\ &= [k_1 + n^*k_4 + n^*[k_5 + n^*(T_{\text{bd}} + T_{\text{asig}}) + (T_{\text{bd}} + T_{\text{asig}})] + k_2 + n^*(T_{\text{bd}} \\ &\quad + T_{\text{asig}}) + k_3 \\ &= k_1 + n^*k_4 + n^*[k_5 + n^*k_6 + k_7] + k_2 + n^*k_8 + k_3 \\ &= k_1 + n^*k_4 + n^*k_5 + n^{2*}k_6 + n^*k_7 + k_2 + n^*k_8 + k_3 \end{aligned}$$

## AnagramaDa 1: konplexutasunaren analisia

Batuketaren eta biderketa konstantearen inbariantza-erregelak aplikatuta:

$$K_1 + n^* K_4 + n^* K_5 + n^2 * K_6 + n^* K_7 + K_2 + n^* K_8 + K_3 \in O(n^2)$$

non  $n$  den sarrerako karaktere-arrayen luzera.

08/09/2022

Datu-Egiturak eta Algoritmoak

37



## Adibidea: Anagrama? (III)

| hitza1 |   | hitza2 |   |
|--------|---|--------|---|
| A      | H | A      | H |
| A      | B | C      | D |
| 0      | 0 | 0      | 0 |
| 2      | 0 | 0      | 0 |
| 0      | 0 | 0      | 0 |

|   |   |   |   |   |   |   |   |       |   |   |   |   |
|---|---|---|---|---|---|---|---|-------|---|---|---|---|
| A | B | C | D | E | F | G | H | ..... | W | X | Y | Z |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ..... | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | ..... | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ..... | 0 | 0 | 0 | 0 |

Egin beharreko eragiketak:

- 26 zero ipini (alfabetoko letra bakoitzeko bat)
- 4 batuketa egin (hitza1)
- 4 kenketa egin (hitza2)
- 26 komprobaketa egin (kasurik okerranean)

38

## AnagramaDa 2: konplexutasunaren analisia

$$T(n) =$$

Datuuen tamaina:

- n, karaktere-arrayen luzera  
(h1.length)  
Prozesua:
  - Hasieratu alfabetoko letren taula
    - 26 aldiz

- h1eko letrak batu
  - n aldiz
- h2ko letrak kendu
  - n aldiz
- Aztertu alfabetoko letren taula eta  
emaitza itzuli
  - T<sub>ret</sub>

08/09/2022

Datu-Egiturak eta Algoritmoak

39

## AnagramaDa 2: konplexutasunaren analisia

$$\begin{aligned} T(n) &= T_{\text{has}} + T_{\text{for1}} + T_{\text{for2}} + T_{\text{buk}} + T_{\text{ret}} \\ &= K_1 + n^*(T_{\text{bd}} + T_{\text{asig}}) + n^*(T_{\text{bd}} + T_{\text{asig}}) + K_2 + K_3 \\ &= K_1 + n^*K_4 + n^*K_5 + K_2 + K_3 \end{aligned}$$

$$T(n) \in O(n)$$

non n den sarrerako karaktere-arrayen luzera.

08/09/2022

Datu-Egiturak eta Algoritmoak

40

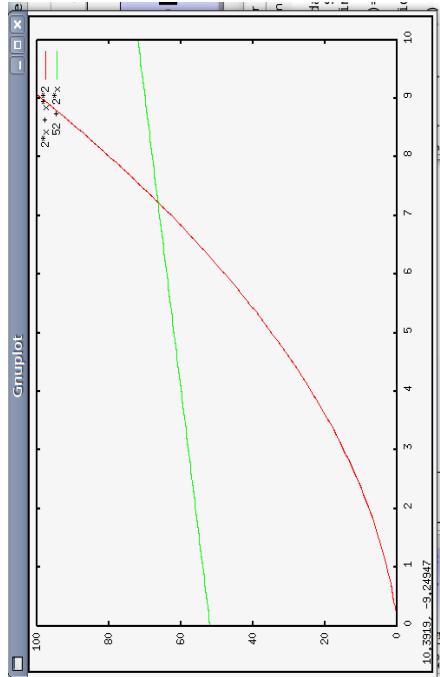
# Algoritmoen konparaketa



Elementuak    **Algoritmo 1**  
4                   $4+4*4+4=24$   
n                   $n+n*n=2n+n^2$

Elementuak    **Algoritmo 2**  
4                   $26+4+4+26=60$   
n                   $26+n+n+26=52+2n$

n handia denean, bigarren algoritmoa lehenengoa baina hobea da.



41

## Funtzioen hazkundeak



| log n | n   | n log n | $n^2$  | $n^3$     | $2^n$           |
|-------|-----|---------|--------|-----------|-----------------|
| 1     | 2   | 2       | 4      | 8         | 4               |
| 2     | 4   | 8       | 16     | 64        | 16              |
| 3     | 8   | 24      | 64     | 512       | 256             |
| 4     | 16  | 64      | 256    | 4.096     | 65.536          |
| 5     | 32  | 160     | 1.024  | 32.768    | 4.294.967.296   |
| 6     | 64  | 384     | 4.096  | 262.144   | $1,8 * 10^{19}$ |
| 7     | 128 | 896     | 16.536 | 2.097.152 | $3,4 * 10^{38}$ |

42



## Konplexutasuna O Notazioa

$7n - 3 \in O(n)$   
 $20n^3 + 10n\log n + 5 \in O(n^3)$   
 $3 \log n + \log \log n \in O(\log n)$   
 $2^{100} \in O(1)$   
 $5/n \in O(1/n)$

43



## Definizioak

$20n^3 + 10n\log n + 5 \in O(n^3)$

Funtzio baten **termino nagusia**, hazkunde-abiadura handiena duen terminoa da

Funtzio baten **hazkunde-abiadura**, bere termino nagusien mendie dago, beste elementu guztiak kontuan hartu gabe.

Funtzio baten **ordena** bere hazkunde-abiadurarekin erlazionatuta dago: termino nagusia soilik aztertu beharko da.

44

## Adibidea I



Array batzen elementu handiena lortu  
Sarrera: n elementuz (osoak) osatutako A array bat  
Irteera: A arrayaren elementu handiena  
Algoritmo: arrayMax(A,n)

```
current=A[0];
for(i=1;i<n;i++)
 if (A[i]>current) current=A[i]
return current
```

45

## Adibidea I Ebazpena



```
current=A[0];
for(i=1;i<n;i++)
 if (A[i]>current) current=A[i]
return current
```

Kasurik onenean =  $2+4*(n-1)+1 = 4n-1$   
Kasurik txarrenean =  $2+6*(n-1)+1 = 6n-3$

46

## Adibidea II



- A array bat izanik, n elementuz (osoak) osatuta
- Kalkulatu B array bat, non:

$$B[i] = \sum_{j=0}^i A[j]$$

47

## Adibidea II Ebazpena I



```
for (i=0;i<n;i++) {
 b=0;
 for(j=0;j<=i;j++)
 b=b+A[j];
 B[i]=b;
}
return B;
```

Pausoak:

Hasieratu eta taula itzuli :O(n)  
i begizta: n aldiz exekutatzen da: O(n)  
j begizta: 1+2+3+...+n aldiz exekutatzen da=  
((1+n)n)/2 = (n + n<sup>2</sup>)/2 : O(n<sup>2</sup>)  
Totala: O(n)+O(n)+O(n<sup>2</sup>) ∈ O(n<sup>2</sup>)

48



## Adibidea ||

### Ebazpena ||

$$B[i] = A[0] + \dots + A[i-1] + A[i]$$

$$B[i] = B[i-1] + A[i]$$

Aurreko terminoan egin ditugun eragiketak erabiltzen  
ditugu algoritmoaren konplexutasuna murrizteko

49



## Adibidea ||

### Ebazpena ||

```
b=0;
B[0]=A[0]
for(i=1;i<n;i++)
 B[i]=B[i-1]+A[i]
return B
```

Eragiketak:

Hasieratu eta taula itzuli: $O(n)$

Hasierako eragiketak: $O(1)$

i begizta: n aldiz exekutatzen da

Guztira:  $O(n)+O(1)+O(n) \in O(n)$  (Ordena lineala)

50



## Beste adibide bat (I). Zenbat batuketa?

```
function Es_Separable (V, inicio, fin) return integer is
begin
 for i in inicio .. fin loop
 izq:= 0;
 for k in inicio .. i - 1 loop izq:= izq+V(k); end loop;
 der:= 0;
 for k in i .. fin loop der:= der+V(k); end loop;
 if izq = der then return i; end if;
 end loop;
 return 0;
end
```

51



## Beste adibide bat (II). Zenbat batuketa?

```
function Suma_Escalonada (V, inicio, fin)
return (integer×integer) is
begin
 maximo:= -∞;
 indice:= inicio-1;
 m:= (inicio + fin)/2;
 for i in inicio .. m loop
 suma:= 0;
 for k in i .. fin loop suma:= suma+V(k); end loop;
 if suma > maximo then
 maximo:= suma;
 indice:= i;
 end if;
 end loop;
 return (maximo, indice);
end
```

52



## Beste adibide bat (III)

```
function Cua (n: natural) return natural is
begin
 i:= 0;
 while i*i <= n loop
 i:= i+1;
 endloop;
 return i-1;
end
```

53



## Konplexutasuna

Analisi asintotikoaren murritzapenak

- Elementu gutxi daudenean ez da egokia
- Analisi asintotikoa goi-estimazio bat da
- **Exekuzio-denbora: programaren batezbesteko denbora, kasurik okerreneko exekuzio-darbora baino askoz txikiagoa izan daiteke**
- Batzuetan kasu okerrena ez da esanguratsua

54



## Konplexutasuna

Analisi asintotikoaren murritzapenak

- Elementu gutxi daudenean ez da egokia
- Analisi asintotikoa goi-estimazio bat da
- **Exekuzio-denbora: programaren batezbesteko denbora, kasurik okerreneko exekuzio-denbora baino askoz txikiagoa izan daiteteke**
- Batzueta kasu okerrena ez da esanguratsua

55



## Bilatze-algoritmoak (I)

- Elementu bat array batean dagoen ala ez  
Sarrera: A Array bat ( $n$  zenbakি osoz osatua)  
eta e osoko bat  
Irteera: true bsb  $e \in A$ ; false, bestela  
Algoritmoa: dago( $A, n, e$ )

```
for(i=0;i<n;i++)
 if (A[i]==e) return true;
return false
```

56



## Bilatze-algoritmoak (II)

Bilaketa dijikotomikoa

Elementu bat array batean dagoen ala ez  
Sarrera: A Array **ordenatu** bat (n zenbakí osoz osatua)

eta e osoko bat

Irteera: true bsb e  $\in A$ ; false, bestela

Algoritmoa: dago(A,n,e)

```
int k=0;
int a=n-1;
while(k<=a) {
 int i=(k+a)/2;
 if (A[i]==e) return true;
 else {
 if (A[i]>e) a=i-1;
 else k=i+1;
 }
}
return false;
```

Zenbat aldiiz exekutatzen da while begizta?

$n, n/2, n/4, \dots, n/2^i$

Kasurik okerrenean:  $2^i = n$

Orduan:  $i=\log n$

$O(\log N)$

57

Bilaketa bitarraren konplexutasun-azterketa

```
int ezker, eskuin, erdiko;
ezker = 0;
eskuin = arr.length - 1;
erdiko = (ezker + eskuin) / 2;

while (ezker < eskuin & arr[erdiko] != x)
{
 if (arr[erdiko]<x)
 ezker = erdiko + 1;
 else //arr[erdiko]>x
 eskuin = erdiko - 1;
 erdiko = (ezker + eskuin) / 2;
}
return (arr[erdiko]==x);
```

$T(n) = T(n) + T_{\text{has}}$

$T_{\text{has}} = T_{\text{ret}}$

$$\begin{aligned}
 T(n) &= T_{\text{has}} + T_{\text{while}} + T_{\text{ret}} \\
 &= K_1 + \log_2 n * T_{\text{if}} + T_{\text{ret}} \\
 &= K_1 + \log_2 n * K_2 + K_3
 \end{aligned}$$

$T(n) \in O(\log_2 n)$

non  $n$  den sarrerako arrayaren luzera.

08/09/2022

Datu-Egiturak eta Algoritmoak

59



## Ordenazio-algoritmoak

Ordenaziorako Java klaseak

public class SortAndSearch <T extends Comparable<T>> {

```
public SortAndSearch() {}
```

```
public void bubbleSort(T[] taula)
public void selectionSort(T[] taula)
public void insertionSort(T[] taula)
public void mergeSort(T[] taula)
public void quickSort(T[] taula)
```

60



## Ordenazio-algoritmoak

### Burbullaren metodoa

```
public void bubbleSort(T[] taula) {
 int out, in;

 for (out = taula.length - 1; out > 0; out--)
 for (in = 0; in < out; in++)
 if (taula[in].compareTo(taula[in + 1]) > 0)
 swap(taula, in, in + 1);
}

private void swap(T[] taula, int one, int two) {
 T temp = taula[one];
 taula[one] = taula[two];
 taula[two] = temp;
}
```

61



## Ordenazio-algoritmoak

### Hautaketa bidezko metodoa

```
public void selectionSort(T[] taula) {
 int out, in, min;

 for (out = 0; out < taula.length - 1; out++)
 {
 min = out; // minimoaren indizea, oraingoz
 for (in = out + 1; in < taula.length; in++)
 if (taula[in].compareTo(taula[min]) < 0)
 min = in; // minimoaren indizea, eguneratua
 swap(taula, out, min);
 }
}
```

62



## Ordenazio-algoritmoak

### Txertaketa bidezko metodoa

```
public void insertionSort(T[] taula){
 int in, out;

 for (out = 1; out < taula.length; out++)
 // out indizaren aurrekoak ordenatuta daude
 {
 T temp = taula[out];
 in = out;
 while (in > 0 && taula[in - 1].compareTo(temp) > 0)
 {
 taula[in] = taula[in - 1]; // (in-1) indizeko elementua eskuinera ekarri
 in--; // indizea ezkerrera ekarri
 }
 taula[in] = temp;
 // temp balioa txertatzen dugu tokatzen zaion posizioan
 }
}
```

63



## Ordenazio-algoritmoak

### Fusioz ordenaketa edo bateratze-metodoa

```
public void mergeSort(T[] taula){
 mergeSort(taula, 0, taula.length-1);
}
```

```
private void mergeSort (T[] taulaBat, int hasiera, int bukaera){
 if (hasiera < bukaera) { // taulan elementu bat baino gehiago badago
 mergeSort(taulaBat, hasiera, (hasiera+bukaera)/2);
 mergeSort(taulaBat, ((hasiera+bukaera)/2)+1, bukaera);
 bateratze(taulaBat, hasiera, (hasiera+bukaera)/2, bukaera);
 }
}
```

64



## Ordenazio-algoritmoak

### Bateratzea, bi taula ordenatuena

```
private void bateratze(T[] taula, int i, int erdikoa, int f){
 T[] bateratua = (T[])new Comparable[f-i+1];
 int ezker = i;
 int eskuin = erdikoa+1;
 int k = 0; //bateratua taula betetzeko indizea
 while (ezker<=erdikoa && eskuin<=f){
 if (taula[ezker].compareTo(taula[eskuin])<= 0){
 bateratua[k] = taula[ezker];
 k++;
 ezker++;
 }
 else {
 bateratua[k] = taula[eskuin];
 k++;
 eskuin++;
 }
 }
}
```

65



## Ordenazio-algoritmoak

### Quicksort

```
public void quickSort(T[] taula){
 quickSort(taula, 0, taula.length-1);
}
```

```
private void quickSort(T[] taulaBat, int hasiera, int bukaera){
 if (bukaera - hasiera> 0) { //taulan elementu bat baino gehiago
 int indizeaZatiketa = zatiketa(taulaBat, hasiera, bukaera);
 quickSort(taulaBat, hasiera, indizeaZatiketa - 1);
 quickSort(taulaBat, indizeaZatiketa + 1, bukaera);
 }
}
```

66



# Ordenazio-algoritmoak

## Quicksort

```
private int zatiketa(T[] taula, int i, int f){
 T lag = taula[i];
 int ezker = i;
 int eskuin = f;

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

 return eskuin;
}
```

67



## Bestelakoak

- Irakurketa gehigarria:
  - [Lewis, Chase 2010], 8. kapitulua
  - Wikipedia (ingelesez). Algoritmoen analisia:  
[http://en.wikipedia.org/wiki/Analysis\\_of\\_algorithms](http://en.wikipedia.org/wiki/Analysis_of_algorithms)
  - Wikipedia (gaztelaniaz). Ordenazio-algoritmoak:  
[http://es.wikipedia.org/wiki/Algoritmo\\_de\\_ordenamiento](http://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento)
- Simulatzialeak:
  - <http://math.hws.edu/eck/js/sorting/xSortLab.html>
  - <http://www.sorting-algorithms.com/>
- Quicksorten dantza:
  - <http://www.youtube.com/watch?v=ywWBy6J5gz8>

68

# Kontu hauetan sakontzeko irakurketa

- [Lewis & Chase 2010]
  - 2. kapitula
- *Algorithms, 4th Edition,*  
Robert Sedgewick and Kevin Wayne  
<http://algs4.cs.princeton.edu/home/>

Sarrera

69

1. **ariketa.** Kalkulatu algoritmo bakoitzaren ordena, matrizearen tamainaren arabera.  
Zein da eraginkorrena?

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b> Matrizea <b>is array</b> (1 .. Lerroak, 1 .. Zutabeak) <b>of</b> Integer;                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <pre> <b>function</b> Batural1 (Mat :      <b>in</b> Matrizea)     <b>return</b> Integer <b>is</b>         Guztira : Integer;         I, J: Integer;     <b>begin</b>         Guztira:= 0;         I := 1;         <b>while</b> (I &lt;= Lerroak) <b>loop</b>             Guztira:=                 Guztira + Mat (I, J);             J := 1;             <b>while</b> (J &lt;= Zutabeak) <b>loop</b>                 Guztira:=                     Guztira + Mat (I, J);                 J := J + 1;             <b>end loop;</b>             I := I + 1;         <b>end loop;</b>         <b>return</b> Guztira;     <b>end</b> Batural1; </pre> | <pre> <b>function</b> Batura2 (Mat: <b>in</b> Matrizea )     <b>return</b> Integer <b>is</b>         Guztira: Integer;         I, J: Integer;     <b>begin</b>         Guztira:= 0;         I := 1; J := 1;         <b>while</b> (I &lt;= Lerroak) <b>loop</b>             Guztira:=                 Guztira + Mat (I, J);             <b>if</b> (J &lt; Zutabeak)                 <b>then</b> J := J + 1;             <b>else</b> J := 1;                 I := I + 1;             <b>end if;</b>         <b>end loop;</b>         <b>return</b> Guztira;     <b>end</b> Batura2; </pre> | <pre> <b>function</b> Errenkadaren_Batura (     Mat : <b>in</b> Matrizea;     Errenkada: <b>in</b> Integer)     <b>return</b> Integer <b>is</b>         Guztira: Integer;     <b>begin</b>         Guztira:= 0;         <b>for</b> J <b>in</b> 1 .. Zutabeak <b>loop</b>             Guztira:= Guztira +                 Mat (Errenkada, J);         <b>end loop;</b>         <b>return</b> Guztira;     <b>end</b> Errenkadaren_Batura;      <b>return</b> Guztira; </pre> |

## **2. ariketa.** Honako algoritmoaren konplexutasun-kostua aztertu:

```
function Muga_Gainditzen (n: Integer) return Integer is

 Jauzia: Integer := 1;
 Posizioa: Integer := 0;
begin
 while Posizioa < n loop
 Posizioa := Jauzia;
 Jauzia := jauzia * 2;
 end loop;
 return Posizioa;
end Muga_Gainditzen;
```

**3. ariketa.** Ikasleen zerrenda bat dugu, eta ikasle bakoitzeko matrikulatu den asignaturen zerrenda.

- Ikasle baten zentzakia emanda, zentbat denbora hartuko du ikasle hori algebran matrikulatuta dagoen jakiteak?
  
- Asignatura bakoitzeko, matrikuluan dituen ikasleen zerrenda lortu nahi da.  
Zein da algoritmo horren kostua?

4. ariketa. Hirugarren kurtsoko ikasle batek algoritmo bat egin du, “primitiva”ko txartelen zerrenda emanda (bakoitzaz dagokion sariarekin) sari handienak dauzkaten 1.000 txartelak lortzeko.

```
type Txartelen_Zerrenda is record
 Taula: array (1 .. Max) of Txartela; -- Txartela mota definitua
 dago
 Zenbat: Integer;
 end record;
```

```
function 1000_Hoberenak (Txartelak : in Txartelen_Zerrenda)
 return Txartelen_Zerrenda is
 -- post: Txartelak zerrendako 1.000 hoberenak lortu dira emaitzan

 Emaitza: Txartelen_Zerrenda;
 begin
 Hasieratu (Emaitza);
 I:= 1;

 while (I <= Txartelak.Zenbat) loop
 if Emaitza.Zenbat < 1000 or
 Saria(Txartelak.Taula(I)) > Saria(Txikiena(Emaitza))
 then sartu txartel hau Emaitza-ren bukaeraan
 Ordenatu (Emaitza); -- burbuilaren bidez
 end if;
 end loop;

 return Emaitza;
 end 1000_Hoberenak;
```

Ondokoak eskatzen da:

- a) Aztertu soluzioa eta bere kostua kalkulatu
- b) Algoritmo hori 14.000.000 txartelen zerrenda batean aplikatzen denean, ikasleak dio 8/9 ordu hartzen dituela, 1.000 txartel hoberenak lortzeko. Lagunduko zenioke algoritmoa hobetzen? Zure erantzuna arrazoitu.

### 5. ariketa Klase hau emanda:

```
public class Liburu {
 int mota;
 // 3 balio posible: 1 (abenturak), 2 (historia) edo 3 (zientzia)
 String izenburua;
 String egileak;
}
```

Liburuen array bat ordenatu nahi da, lehenengo abenturazkoak egoteko, ondoren historiakoak eta azkenean zientziakoak. Kategoria bakoitzaren barruan, ordenak ez du garrantzirik. 2 algoritmo eman dizkigute:

#### Algoritmo 1

```
public void bubbleSort(Liburu[] taula) {
 int out, in;

 for (out = taula.length - 1; out > 0; out--)
 for (in = 0; in < out; in++)
 if (taula[in].mota > taula[in + 1].mota)
 swap(taula, in, in + 1);
}
```

#### Algoritmo 2

```
public void misterio(Liburu[] taula) {
 int in, i1, i2, i3 = 0;
 Liburu[] t1, t2, t3; // 3 array lagungarriri
 t1 = new Liburu[taula.length];
 t2 = new Liburu[taula.length];
 t3 = new Liburu[taula.length];

 for (in = 0; in < taula.length; in++)
 if (taula[in].mota == 1) { t1[i1] = taula[in]; i1++; }
 else if (taula[in].mota == 2) { t2[i2] = taula[in]; i2++; }
 else if (taula[in].mota == 3) { t3[i3] = taula[in]; i3++; }

 int i = 0;
 for (in = 0; in < i1; in++) { taula[i] = t1[in]; i++; }
 for (in = 0; in < i2; in++) { taula[i] = t2[in]; i++; }
 for (in = 0; in < i3; in++) { taula[i] = t3[in]; i++; }
}
```

Hau eskatzen da:

- Azaldu zer egiten duen algoritmo bakoitzak
- Esan, modu arrazoituan, zein den algoritmo bakoitzaren kostua, eta horren arabera, 2 algoritmoetatik zein izango litzatekeen eraginkorrena

6. ariketa Bi algoritmo hauek emanda:

### Algoritmo 1

```
public static int count(int[] a) {
 // a[i] guztiaik != 0 eta desberdinak
 int N = a.length;

 int cnt = 0;

 for (int i = 0; i < N; i++) {
 for (int j = i+1; j < N; j++) {
 if (a[i] + a[j] == 0) {
 cnt++;
 }
 }
 }

 return cnt;
}
```

### Algoritmo 2

```
public static int count(int[] a) {
 // a[i] guztiaik != 0 eta desberdinak
 int N = a.length;

 Arrays.sort(a); // mergesort

 int cnt = 0;

 for (int i = 0; i < N; i++) {
 int j = Arrays.binarySearch(a, -a[i]); // bilaketa bitarra
 if (j > i) cnt++;
 }

 return cnt;
}
```

```
public int binarySearch(int[] a, int x)
// Aurrebaldintza:
// Postbaldintza: x duen elementuaren posizioa bueltatzen du
// 0 bueltatuko da x ez badago a arrayan
```

Hu eskatzen da:

- a) Esan zer egiten duen algoritmo bakotzak
- b) Bereain kostua kalkulatu era arrazoituan

### Balioak ordenatu (0,5 puntu)

Ikasle batek algoritmo hau idatzi du zerranda bateko balioak ordenatzeko:

```
public ArrayList<String> ordenatu (ArrayList<String> l) {
 // post: emaitza sarrerako zerrenda izango da, elementuak
 // ordenatuta dituelarik

 ArrayList<String> emaitza = new ArrayList<String>();

 Iterator<String> it = l.iterator();

 while (it.hasNext ()) {
 String s = it.next ();
 emaitza.addLast (s);
 emaitza.quicksort ();
 }

 return emaitza;
}
```

a) Kalkulatu algoritmo horren kostua, **modu arrazoituan**.

b) Esan ea problema hori ebazteko soluzio eraginkorrago bat ikusten duzun. Baiezko kasuan, idatzi soluzio berri horren kodea eta esan zein den bere kostua.

Bi zenbaki positiboak emanda x eta n, esan zer kalklatzen duen ondoko metodoak. Zein da bere kostua? Erantzuna arrazoitu.

```
public static Integer exponentzial(Integer x, Integer n)
{
 Integer resultado; Integer exp;

 if n == 0 return 1;
 else { resultado = x;
 exp = 1;

 while (2 * exp < n)
 resultado = resultado * resultado;
 exp = 2 * exp;
 }
 return resultado
}
```

## *Listari buelta eman*

Bi algoritmo ditugu zerrenda bateko elementuei buelta emateko:

a)

```
public ArrayList<T> invertir(ArrayList<T> l) {
 // post: el resultado contiene los elementos de la lista original en
 // orden inverso
 ArrayList<T> resultado = new ArrayList<T>();
 Iterator<T> it = l.iterator();
 while it.hasNext() {
 T s = it.next();
 resultado.addFirst(s);
 }
 return resultado;
}
```

b)

```
public ArrayList<T> invertir(ArrayList<T> l) {
 // post: el resultado contiene los elementos de la lista original en
 // orden inverso
 ArrayList<T> resultado = new ArrayList<T>();
 Stack<T> aux = new Stack<T>();

 Iterator<T> it = l.iterator();

 while it.hasNext() {
 T s = it.next();
 aux.push(s);
 }

 while (!aux.isEmpty()) {
 T s = aux.pop();
 resultado.addLast(s);
 }
 return resultado;
}
```

a) Kalkulatu algoritmoen kostua, **modu arrazoituan**

b) Esan zein soluzio den eraginkorrena

## 1. (0,5 puntu) Bi algoritmo ditugu:

```
public ArrayList<Integer> ebakidura1(Integer[] a, Integer[] b) {
 // Aurrebaldeintza: a-k ez du balio errepikaturik
 // b-k ez du balio errepikaturik

 ArrayList<Integer> emaitza = new ArrayList<Integer>();

 for (int i = 0; i < a.length; i++) {
 for (int j = 0; j < b.length; j++) {
 if (a[i] == b[j]) {
 emaitza.add(a[i]);
 }
 }
 }

 return emaitza;
}
```

```
public ArrayList<Integer> ebakidura2(Integer[] a, Integer[] b) {
 // Aurrebaldeintza: a-k ez du balio errepikaturik
 // b-k ez du balio errepikaturik

 Arrays.sort(a); // mergesort
 Arrays.sort(b);

 ArrayList<Integer> emaitza = new ArrayList<Integer>();

 int i = 0;
 int j = 0;
 while ((i < a.length) && (j < b.length)) {
 if (a[i] == b[j]) { emaitza.add(a[i]); i++; j++; }
 else if (a[i] < b[j]) i++;
 else j++;
 }

 return emaitza;
}
```

Hau eskatzen da:

- Azaldu zer egiten duen algoritmo bakoitzak
- Esan, modu arrazoiutan, zein den algoritmo eraginkorrena, algoritmoen kostuan oinarrituz.

## 1. (0,5 puntu) hiru algoritmo ditugu:

```
public void urrunenDaudenak1(Integer[] a) {
 // Aurrebaldeintza: array-ak bakarrik balio positiboak ditu

 Integer max = -1;
 Integer indicei;
 Integer indicej;

 for (int i = 0; i < a.length; i++) {
 for (int j = i + 1; j < a.length; j++) {
 if (max < balioAbsolutua(a[i] - a[j])) {
 max = balioAbsolutua(a[i] - a[j]);
 indicei = i;
 indicej = j;
 }
 }
 System.out.println("bi balio urrunenak " + a[indicei] + " " + a[indicej] + " dira");
 }
}
```

```
public void urrunenDaudenak2(Integer[] a) {

 Arrays.sort(a); // mergesort

 System.out.println(" bi balio urrunenak " + a[0] + " " + a[a.length - 1] + " dira");
}
```

```
public void urrunenDaudenak3(Integer[] a) {

 Integer indicei = maximoAbilatu(a);
 Integer indicej = minimoAbilatu(a);

 System.out.println("bi balio urrunenak " + a[indicei] + " " + a[indicej] + " dira");
}
```

Hau eskatzen da:

- Azaldu zer egiten duen algoritmo bakoitzak
- Esan, modu arrazoituan, zein den algoritmo eraginkorrena, algoritmoen kostuan oinarrituz.

## 2. gaia: Oinarrizko datu-egitura linealak

2022/09/08

Datu-Egiturak eta Algoritmoak

1

## 2. gaia: Oinarrizko datu-egitura linealak

### 2.0. Sarrera.

#### 2.0.1. Array egiturak

- 2.0.2. Egitura estekatuak
- 2.0.3. DMAK. Interfazeak. Klase generikoak
- 2.0.4. Iteradoreak
  - 2.1. Pilak
  - 2.2. Ilarak
  - 2.3. Listak (zerrendak)

2022/09/08

Datu-Egiturak eta Algoritmoak

2

## Datu-egiturak

- Bi alderdi bereizten dira:
  - Datuen errepresentazioa eta antolamendua
    - Oinarrizko elementuen mota eta beren antolaketa
  - Egitura horien gaineko eragiketak
    - Eragiketa horiek dira datu-egitura erabiltzeko eskura ditugunak
    - Ohiko eragiketak:
      - Atzipena
      - Bilaketa
      - Txertaketa
      - Ezabaketa

2022/09/08

Datu-Egiturak eta Algoritmoak

3

## Datu-egiturak, objektuei orientatuta

- Objektuei orientatutako paradigmari, datu-egiturak klaseten definitzen dira:
  - Badira aurredefinitorako klaseak
    - Adibidez, java.util.paketean daudenak
      - java.util.ArrayList
      - java.util.Stack
  - Programatzaleak defini ditzake bere neurrikoak

2022/09/08

Datu-Egiturak eta Algoritmoak

4

## Datu-egituren ezaugarri zentzitakoak, baliagarriak izan daitezzen

- Generikotasuna: oinarrizko elementuen mota parametriza daiteke
  - public class ArrayList<E> extends AbstractList<E>{  
...  
}
  - Berrerabilpena: programatutako datu-egiturak beste programetan erabil daitezke
    - import java.util.ArrayList;
    - import java.util.Queue;
    - ...  
ArrayList<String> izenak = new ArrayList<String>();

2022/09/08

Datu-Egiturak eta Algoritmoeak

5

## Abstrakzioa: interfazea vs implementazioa

- Funtzionalitatearen definizioa (interfazea) ez dago implementazioaren menpe
- Adibidez, Queue interfazea:

public interface Queue<E> extends Collection<E>

### Summary of Queue methods

|         | Throws exception | Returns special value |
|---------|------------------|-----------------------|
| Insert  | add(e)           | offer(e)              |
| Remove  | remove()         | poll()                |
| Examine | element()        | peek()                |

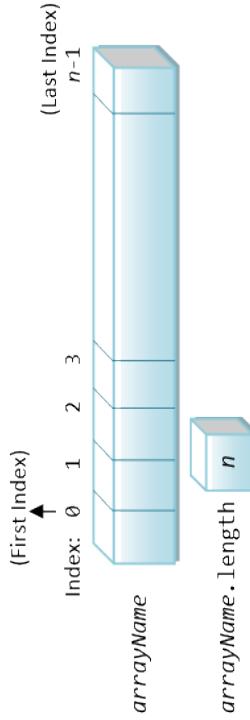
2022/09/08

Datu-Egiturak eta Algoritmoeak

6

## 2.0.1 Arraya edo taula

- Mota bereko elementuen bildumak dira
- Atzipena indexazio-eragilearen bidez egiten da: `[ ]`
- Oinarritzko mota:
  - Primitiboa
  - Erreferentzia (objektua)
- Oso eraginkorrik
- Oso zurrunkak



2022/09/08

Datu-Egiturak eta Algoritmoak

7

## Arrayak

- Erazagupena:
  - mota `arrayIzena[]`;
  - mota `[ ] arrayIzena`;
- Sorkuntza
  - Elementuen kopurua adierazita, memoriako espazioa **ereserbatzen da**
    - `arrayIzena = new mota[tamaina];`
  - Hasieratu bitartean, arrayaren elementuek beren motako **balio lehenetsiak** hartzen dituzte
    - Hasieraketa

**Balio lehenetsiak:**  
int, short, long = 0  
float, double = 0.0  
booleanrak = false  
String = null  
Object = null

2022/09/08

Datu-Egiturak eta Algoritmoak

8

## Arrayak

- Atributu bakarra: **final int length**
- Erazagutu, sortu eta hasieratu egin behar dira

```
int[] taula;

taula = new int[3];

for (int i=0; i<3; i++)
 taula[i] = i+1;

• edo:

int[] t = {1, 2, 3};
```

## Dimentsio anitzeko arrayak

- char a[][]; //Erazagutu
  - a = new char[3][3] //Sortu
  - a[0][0]='A'; //Hasieratu
  - ...
- A[0][2]='C'

|   |   |   |   |
|---|---|---|---|
|   | 0 | 1 | 2 |
| 0 | A | B | C |
| 1 | D | E | F |
| 2 | G | H | I |

## Accessing Elements of Two-Dimensional Arrays

Elements in two-dimensional arrays are commonly referred by `x[i][j]` where 'i' is the row number and 'j' is the column number.

### Syntax:

```
x[row_index][column_index]
```

For example:

```
int[][] arr = new int[10][20];
arr[0][0] = 1;
```

2022/09/08

11

## Taulen mugak eta bestelako aukerak

- Taulak ez dira oso malguak
- Antolakuntza lineala mantendu, baina manipulatzeko ahalmena aberasteko beharra izaten da batzuetan:
  - Tamaina aldatu exekuzio-aldean
    - Lekualdatu, ordenatu, etab.
  - Alternatiba bat: **listak**
    - Era eta molde askotakoak izan daitezke

2022/09/08

Datu-Egiturak eta Algoritmoak

12

## **Lista batek izan ditzakeen eragiketak**

- Eraiki:
  - Lista hutsa sortu
  - Elementu bat gehitu
- Informazioa:
  - Hutsik dago?
  - Zer luzera du?
- Atzipen lokala:
  - Balioa atzitu
  - Balioa egeneratu
  - Balioa kokatu
- Osaketa eta aldaketak
  - Elementuak ertzetatik erantsi/kendu
  - Elementua txertatu posizioan
  - Elementua ezabatu posiziotik
  - Elementu jakin bat ezabatu

2022/09/08

Datu-Egiturak eta Algoritmoak

13

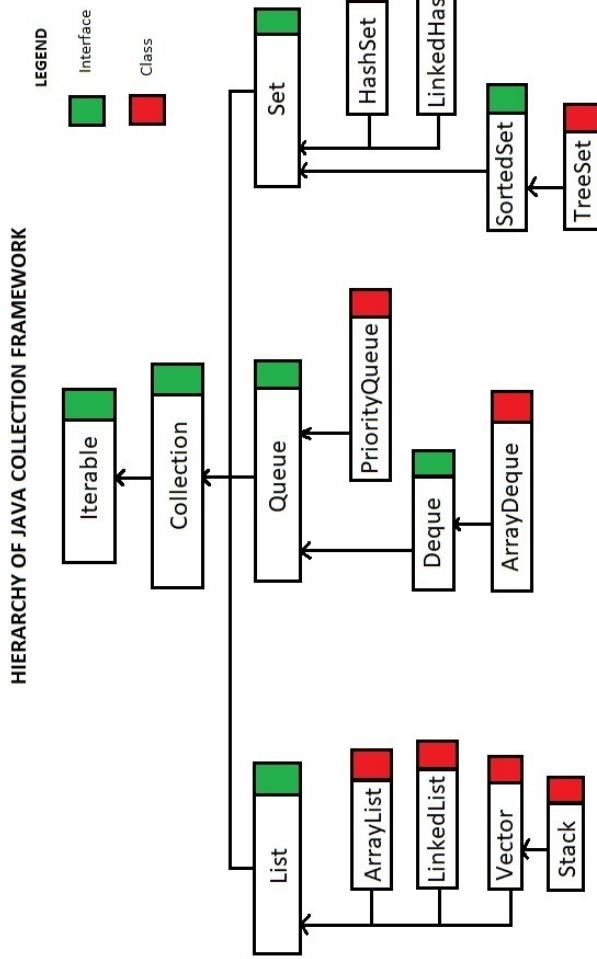
## **Java Collections**

- Landu edo garatuko ditugun egitura gehienek bertsio estandarra badute Javan (`java.util`)
- Horiek dira gehien erabiltzen direnak mundu errealean

2022/09/08

Datu-Egiturak eta Algoritmoak

14



## Bektoreak. Vector klasea

- Datu mota bereko elementuen bilduma bat da bektore bat
- Bektorearen tamaina handitu eta txikitu egin daiteke **dinamikoki**
- Elementuak **indize** baten bidez atzi daitezke, baina **ez [ ] bidez**
- Bektorearen tamaina:
  - **capacity()** metodoak bektoreak har ditzakeen elementuen kopurua itzultzen du
  - **size()** metodoak une horretan dauzkan elementuen kopuruua itzultzen du
  - **capacityIncrement()** aldagaiak adierazten du zenbat haziko den bektorea, hazteko beharra daukanean

## Vector klasea

- Elementu multzo bat kudeatzeko erabiltzen da
- `java.util` paketean dago eskuragarri
  - <https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html>

| Eraikitzailreak                                                    | Esanahia                                                                                                                                    |
|--------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Vector&lt;T&gt;()</code>                                     | Bektore huts bat sortzen du                                                                                                                 |
| <code>Vector&lt;T&gt;(int capacity)</code>                         | <i>capacity</i> kopuruko bektore huts bat sortzen du                                                                                        |
| <code>Vector&lt;T&gt; (int capacity, int capacityIncrement)</code> | <i>capacity</i> kopuruko bektore bat sortzen du<br>eta bere gehikuntza kopurua<br><i>capacity</i> / <i>increment</i> aldagaien finkatzen da |

2022/09/08

17

## Vector klasea. Atzipen-metodoak

| Atzipen-metodoak                             | Esanahia                                      |
|----------------------------------------------|-----------------------------------------------|
| <code>T firstElement()</code>                | Lehenengo elementua itzultzen du              |
| <code>T lastElement()</code>                 | Azkeneko elementua itzultzen du               |
| <code>T elementAt(int index)</code>          | Index posizioan dagoen elementua itzultzen du |
| <code>Enumeration&lt;T&gt; elements()</code> | Elementuen zerrenda itzultzen du              |

2022/09/08

18

Datu-Egiturak eta Algoritmoak

## Vector klasea. Atzipen-metodoak

|                                    |                                                                                                         |
|------------------------------------|---------------------------------------------------------------------------------------------------------|
| boolean contains(T elem)           | elem elementua bektoarean dagoen ala ez izultzen du                                                     |
| int indexOf(T elem)                | elem elementua bektoarean lehenengo<br>aldiz agertzen den posizioa itzultzen du                         |
| int indexOf(T elem, int index)     | elem elementua bektoarean index posiziotik aurera<br>lehenengo aldiz agertzen den posizioa itzultzen du |
| int lastIndexOf(T elem)            | elem elementua bektoarean agertzen den<br>azken adiaren posizioa itzultzen du                           |
| int lastIndexOf(T elem, int index) | elem elementua bektoarean index posiziotik atzera<br>lehenengo aldiz agertzen den posizioa itzultzen du |

2022/09/08

Datu-Egiturak eta Algoritmoak

19

## Vector klasea. Txertatu, ezabatu eta aldatzeko metodoak

| <b>Elementuak txertatu, ezabatu eta<br/>aldatzeko metodoak</b> | <b>Esanahia</b>                                                      |
|----------------------------------------------------------------|----------------------------------------------------------------------|
| void add(T elem)                                               | elem objektua txertatzen da bektore bultaeran                        |
| void insertElementAt(T elem, int index)                        | elem objektua txertatzen da index posizioan                          |
| void setElementAt(T elem, int index)                           | index posizioan dagoen elementua<br>elem objektarekin ordezkatzen da |
| boolean removeElement(T elem)                                  | elem objektuaren lehen agerpena ezabatzen da                         |
| void removeElementAt(int index)                                | index posiziko elementua ezabatzen da                                |
| void removeAllElements()                                       | elementu guztia ezabatzen dira                                       |

2022/09/08

Datu-Egiturak eta Algoritmoak

20

## Vector klasea. Tomainari dagozkion metodoak

| Metodoak                  | Esanahia                                                     |
|---------------------------|--------------------------------------------------------------|
| int capacity()            | bektoreak har dezakeen elementu kopurua itzultzen du         |
| int size()                | une horretan dauzkan elementu kopurua itzultzen du           |
| void setSize(int newSize) | Tamaina berri bat finkatzen du                               |
| void trimToSize()         | Bektorearen edukiera = bektorearen elementu kopurua          |
| boolean isEmpty()         | Bektorea hutsik badago, true bueltatzten du; false, bestela. |

Vector <Integer> v      

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
|---|---|---|

      v.size() = 3  
                                  v.capacity() = 4

2022/09/08

Datu-Egiturak eta Algoritmoak

21

## Vector klasea.

Erazagupena, sorkuntza, hasieraketa, atzipena

```
import java.util.Vector;
public class ProbaBektoreak{
 public static void main(String args[]){
 Vector<Integer> v;
 v= new Vector<Integer>();
 Integer obj1= new Integer(1);
 Integer obj2= new Integer(2);
 v.add(obj1);
 v.add(obj2);
 Integer i1 = v.elementAt(0);
 int osoa1 = i1.intValue();
 Integer i2 = v.elementAt(1);
 int osoa2 = i2.intValue();
 System.out.println(osoa1+"" + osoa2+" = "+(osoa1+osoa2));
 }
}
```

2022/09/08

Datu-Egiturak eta Algoritmoak

22

|                                         | <b>Arraya</b>                                                         | <b>Bektorea</b>                                                                                                   |
|-----------------------------------------|-----------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| <b>Elementuen mota</b>                  | Erazagutzean ( <i>int a[]</i> )                                       | T datu motako objektuak ( <i>Vector&lt;T&gt; v;</i> )                                                             |
| <b>Tamaina</b>                          | <b>length</b> attributua: <i>a.length</i>                             | <i>size()</i> eta <i>capacity()</i> metodoen bidez ( <i>v.size()</i> )                                            |
| <b>Dinamikoa / Estatikoa</b>            | Sortzean definitzen da tamaina, eta finkoa da ( <i>a = new a[3]</i> ) | Behar adina handitu daiteke <b>dinamikoki</b> ( <i>v.add()</i> )                                                  |
| <b>Esleipena/ Txertaketa/ Ezabaketa</b> | [ ] eragilearen bidez ( <i>a[0]=2</i> )                               | Metodoen bidez: <i>addElement(T obj)</i> , <i>removeElement(T obj)</i> , <i>insertElementAt(T obj, int index)</i> |
| <b>Non dator?</b>                       | java.lang paketean                                                    | java.util paketean                                                                                                |

2022/09/08

Datu-Egiturak eta Algoritmoak

23

## Ariketa

- ArrayList klasea implementatu, arrayen bidez:

```
public class NireArrayList<T> {
 // atributuak
```

```
public NireArrayList() { // eraikitzailea
```

```
}
```

```
public NireArrayList(int tamaina) { // eraikitzailea
```

```
}
```

```
public T get(int i) {
```

```
}
```

24

# Ariketa

```
public void set(int i, T elem) {
}
}
public void add(T elem) {
}
}
public T remove(int index){
}
}
public boolean contains(T elem){
}
}
public int size(){
}
}
```

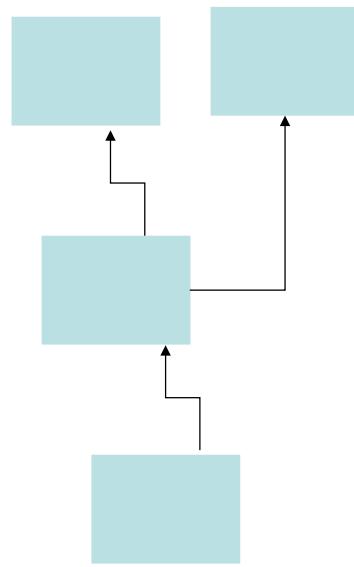
## 2.0.2. Egitura estekatuak

Creative Commons lizenziapean



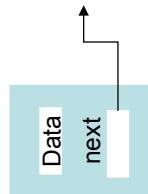
### Egitura estekatu baten ezaugarrriak

- Datu-egiturak dira, objektuen erreferentzia-aldagaiak erabiltzen dituzte beste objektuen estekak izan ahal izateko



## Egitura estekatu baten diseinua

- Orokorrean, **adabegien klase bat** egongo da. Adabegi batek hau izango du:
  - Datuak eta
  - Adabegi klasearen erreferentzia bat edo gehiago (definizio errekurtsiboa)



Egitura estekatuak

3

## Lehen hurbilpena(I)

```
public class Pertsona {
 private String name;
 private String na;
 private Pertsona next; // hurrengo pertsonaren atzipena!
```

```
public Pertsona(String pName, String pNa) { // Eraikitzalea
 name = pName;
 na = pNa;
 next = null;
}
```

```
public void setNext(Pertsona next) { this.next = next; }
```

```
public void print() { // Dena idazten du
```

Egitura estekatuak

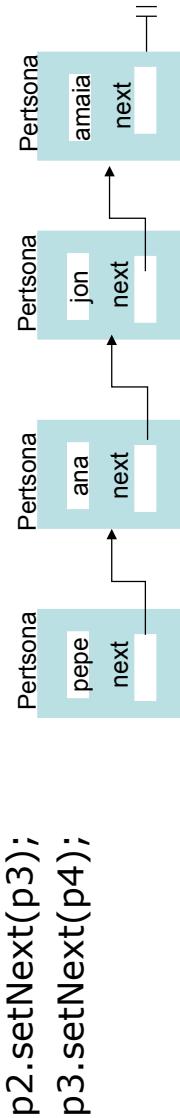
4

## Lehen hurbilpena (II)

```
public static void main(String[] args) {

 Pertsona p1 = new Pertsona("pepe", "1111");
 Pertsona p2 = new Pertsona("ana", "2222");
 Pertsona p3 = new Pertsona("jon", "33333");
 Pertsona p4 = new Pertsona("amaia", "1212");

 p1.setNext(p2);
 p2.setNext(p3);
 p3.setNext(p4);
}
```



Egitura estekatuak

5

## Lehen hurbilpena(III)

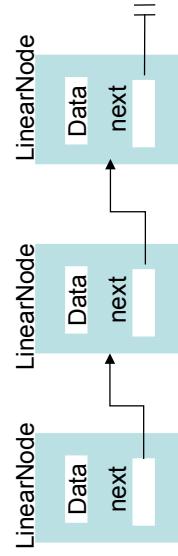
- Pertsona klasea,
  - Pertsona bat da?
  - Edo pertsona-multzo bat?
  - Zer idazten du print-ek? Zer idatzi beharko luke?
- Aurreko implementazioa badabil, baina diseinu ez egokia du
  - Pertsona klasea eta PersonenZerrenda klaseak desberdinak dira

Egitura estekatuak

6

## Adabegi baten definizioa

```
public class LinearNode
{
 public Persona data; // datuak adabegian
 public LinearNode next; // hurrengo adabegia
}
```



Egitura estekatuak

7

## Egitura estekatuaren definizioa

```
class LinkedList
{
 private LinearNode first; // adabegi berezi baten erreferentzia
 // aukerazko atributuak: int size, ...

 public LinkedList() { // eraikitzalea
 first = null;
 }

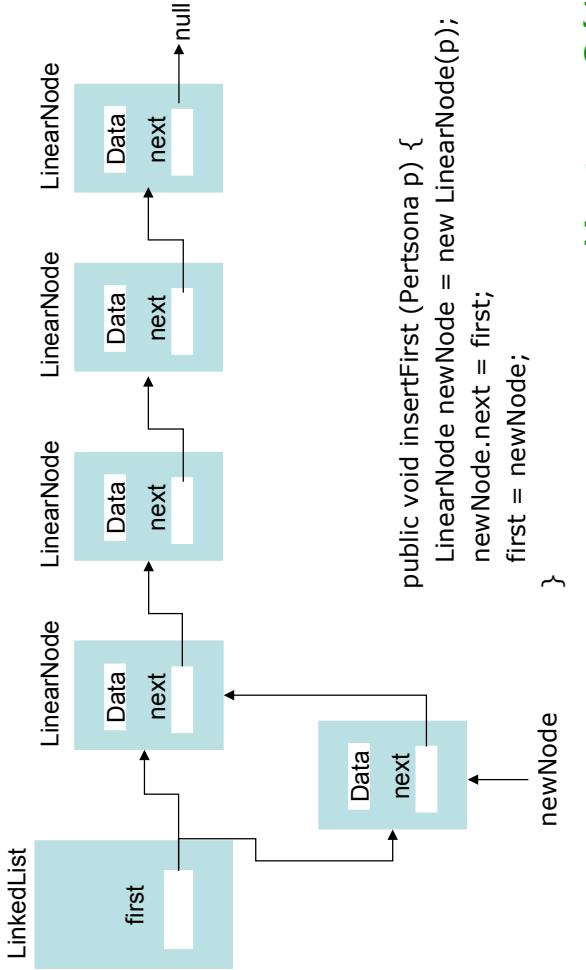
 public boolean isEmpty() {.....}
 public void insertFirst(Persona p) {.....}
 public Persona deleteFirst() {.....}
 public void displayList() {.....}
 public Persona find(Persona p) {.....}
 public Persona delete(Persona p) {.....}
}
```



Egitura estekatuak

8

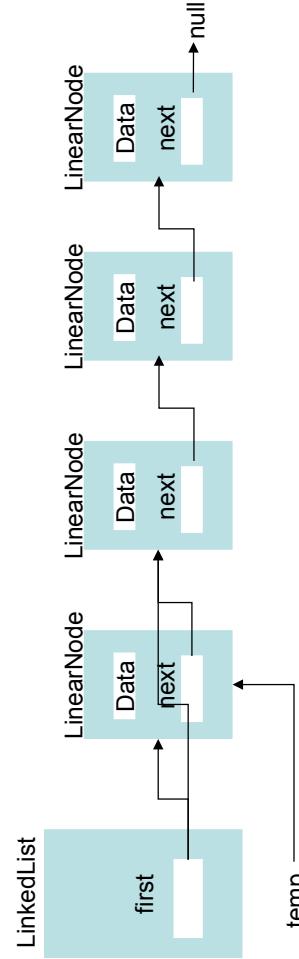
# Txertaketetan hasieran



Egitura estekatuak

9

# Lehenengoa ezabatu



Egitura estekatuak

10

## Egiturako datuak inprimatu

```
public void displayList()
{
 System.out.print("List (first-->last): ");
 LinearNode current = first; // start at beginning of list
 while(current != null) // until end of list,
 {
 current.displayLink(); // print data
 current = current.next; // move to next node
 }
 System.out.println("");
}
```

**Kostua:**  $O(n)$

**n:** egiturako adabegi-kopuruoa

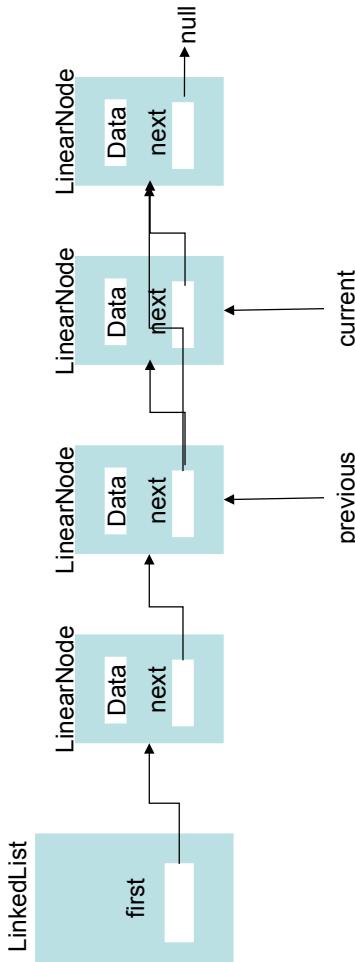
- **Ikusi** Link.java, LinkList.java eta LinkListApp.java

## Bilatu balio jakin bateko adabegia

```
public Pertsona find(Pertsona p) // find node with given key
{
 LinearNode current = first; // (assumes non-empty list)
 while(!current.data.equals(p)) // start at 'first'
 {
 if(current.next == null) // if end of list,
 return null; // didn't find it
 else
 current = current.next; // go to next link
 }
 return current.data; // found it
}
```

**Kostua:**  $O(n)$

## Ezabatu balio jakin bateko adabegia



Egitura estekatuak

13

## Ezabatu balio jakin bateko adabegia

```
public Persona delete(Persona p) // delete link with given key
{
 // Precondition: there exists an element with the given key
 // (assumes non-empty list)
```

```
 ListNode current = first; // search for link
 ListNode previous = first;

 while (!current.data.equals(p)) {
 if (current.next == null)
 return null; // didn't find it
 else
 {
 previous = current; // go to next link
 current = current.next;
 }
 }
 if (current == first) // if first link,
 first = first.next; // change first
 else
 previous.next = current.next; // otherwise,
 // bypass it
 return current.data;
}
```

Egitura estekatuak

14

**Kostua: O(n)**

# Iteradore bat eskaintzeko?

```
/** Return an iterator to the stack that iterates through the items */
public Iterator iterator() { return new ListIterator(); }

// an iterator, doesn't implement remove() since it's optional
private class ListIterator implements Iterator {

 private LinearNode current = first;

 public boolean hasNext() { return current != null; }

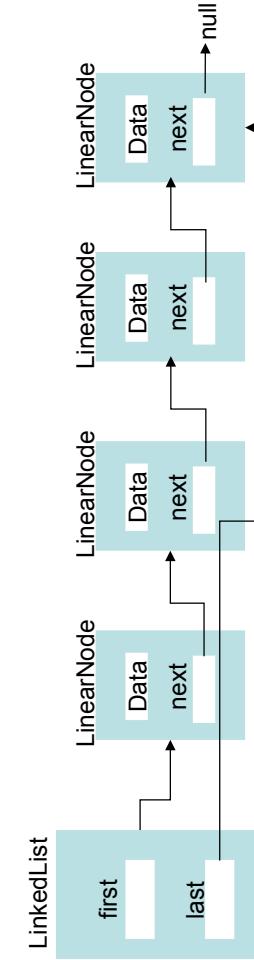
 public void remove() {
 throw new UnsupportedOperationException();
 }

 public Person next() { // recorre el campo clave
 if (!hasNext()) throw new NoSuchElementException();
 Person item = current.Data;
 current = current.next;
 return item;
 }
} // private class
```

Egitura estekatuak

15

## Bulkaeran txertatzeko, komeni da azkenaren atzipena izatea



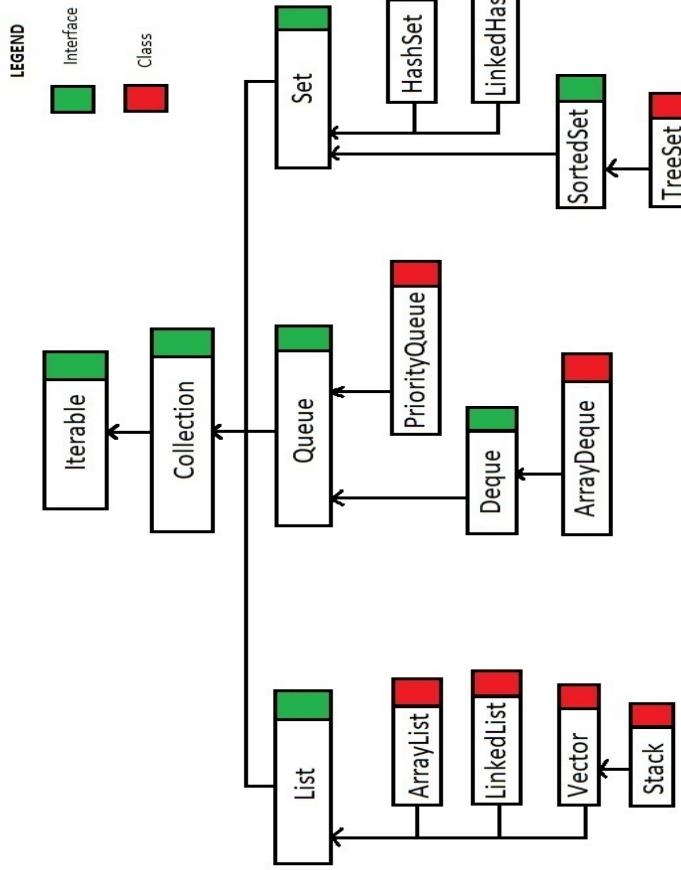
- Ikusi [FirstLastList.java](#)

- 4. kapitulua aztertu: **Linked structures**  
[Lewis eta Chase 2010]

Egitura estekatuak

16

## HIERARCHY OF JAVA COLLECTION FRAMEWORK



09/03/2021

Datu-Egiturak eta Algoritmoak

17

## ArrayList klasea

- *ArrayList* erabiltzen ditu elementuak gordetzeko
  - Txertatze-ordenari eusten dio
  - Elementuen bikoizketa onartzen du
  - Elementuen zuzeneko atzipena (indize bidezkoa)

09/03/2021

Datu-Egiturak eta Algoritmoak

18

## LinkedList klasea

- *Estekadura bikoitzeko listak* erabiltzen ditu elementuak gordetzeko
  - Elementuen bikoizketa onartzen du
  - Elementuen atzipen sekuentziala

09/03/2021

Datu-Egiturak eta Algoritmoak

19



## ArrayList vs LinkedList

|    |   |    |   |    |
|----|---|----|---|----|
| 0  | 1 | 2  | 3 | 4  |
| 23 | 3 | 17 | 9 | 42 |



ArrayList (top) vs LinkedList (bottom). Source [https://dzone.com/storage/temp/895349\\_arraylist-linkedlistt.png](https://dzone.com/storage/temp/895349-arraylist-linkedlistt.png)

09/03/2021

Datu-Egiturak eta Algoritmoak

20

## ArrayList edo LinkedList?

- ArrayList
  - Egokiagoa da elementuen atzipenerako; alegia, get edo set moduko eragiketetan
  - Egokiagoa da elementuen eransketetarako edo ezabaketetarako

|            |                               |    |
|------------|-------------------------------|----|
| 09/03/2021 | Datu-Egiturak eta Algoritmoak | 21 |
|------------|-------------------------------|----|

## ArrayList vs LinkedList . Eragiketen eraginkortasuna

| Function    | Description                                                             | Average Case |            |             |                                                               |
|-------------|-------------------------------------------------------------------------|--------------|------------|-------------|---------------------------------------------------------------|
|             |                                                                         | ArrayList    | LinkedList |             |                                                               |
| get         | Getelement at specified position (in the middle)                        | O(1)         | O(N)       | insertLast  | Inserts the specified element to the end of the list.         |
| getFirst    | Get the first element                                                   | O(1)         | O(1)       | delete      | Removes the element at the specified position (in the middle) |
| getLast     | Get the last element                                                    | O(1)         | O(1)       | deleteFirst | Removes the first element                                     |
| insert      | Inserts the specified element at the specified position (in the middle) | O(N)         | O(N)       | deleteLast  | Removes the last elements                                     |
| insertFirst | Inserts the specified element at the first position                     | O(N)         | O(1)       |             |                                                               |

|            |                               |    |
|------------|-------------------------------|----|
| 09/03/2021 | Datu-Egiturak eta Algoritmoak | 22 |
|------------|-------------------------------|----|

---

DEA

## 2.0.3. DMAak. Interfazeak eta klase generikoak

Creative Commons lizenziapean publikatua  CC BY NC SA

---

EDA

## Programazio rako tresnak

- Orokorrean, egitura bat erabiltzeko, **eragiketa-multzo txiki batera** mugatzear **seguruago** (zuzentasunean) egiten du bere erabilera

## “Mundu errealarren” moldaketa

- Problema bat ebazteko objektuak/kontzeptuak erabiltzen ditugunean, normalean **eragiketa-multzo txiki bat nahiko izaten da**. Interesgarria da jakitea zein den multzo hori kasu bakoitzean

---

3

## Datu mota abstraktua (DMA)

- DMA: **datu multzo baten** eta datu horiekin egin daitezkeen **eragiketa multzo baten** zehaztapena (espezifikazioa) da
- **Abstraktua:** indarra egin daitezkeen **zein** eragiketetan jartzen da, **nola** egin daitekeen kontuan izan gabe
- DMA bat independentea da bere implementazio desberdinakiko

---

4

## Adibidea: kontagailu DMA

**mota** **kontagailu** *(Gogoratu: programazioaren metodologia)*

**erabiltzen du nat**

**eragiketak**

hutsa: → kontagailu *(eraikitzalea)*

inkr: kontagailu → kontagailu *(eraikitzalea)*

dekr : kontagailu → kontagailu *(modifikatzalea)*

reset : kontagailu → kontagailu *(modif.)*

balioa: kontagailu → nat *(konsultakoa)*

5

## ekuazioak

*(Gogoratu: programazioaren metodologia)*

(1) dekr(hutsa) = **erretea** *(errore ekuazioa)*

(2) dekr (inkr (x)) = x

(3) reset(hutsa) = hutsa

(4) reset(inkr(x)) = hutsa

(5) balioa(hutsa) = zero

(6) balioa(inkr(x)) = suc(balioa(x))

Ez-erakitzazile bakotza  
erakitzazile bakotzeo

6

## DMAak diseinu-tresna bezala

- **Zein** eragiketa behar ditut nire aplikazioko datuak erabiltzeko?
  - Sartutako azkenaren atzipena
  - Lehentasun handienekoaren atzipena
  - Lehenengoa **nola** implementatuko diren pentzatu behar da kendu
  - Bi elementu konbinatu
  - ...
- **Ondoren**, eragiketa horiek
  - Implementazio bat aldatu ahal izango da, DMAa erabiltzen ari den aplikazioa aldatu gabe
- Implementaziorako aukerak **eraginkortasun-irizpideek** gidatuta egongo dira

EDA

7

## DMA baten espezifikazioa Javaz

- Softwarearen garapenerako ingurune batean, DMA baten espezifikazioari **DMAaren interfazea** esaten zaio
- Javaz, **interface** eraikuntza konstante eta metodo abstraktuuen bilduma da
  - Metodo abstraktu batean bere signatura (goiburuko) erazagutuko da, baina ez bere implementazioa
- **interface** eraikuntzak DMAak zehazteko balio dezake

EDA

8

## Adibidea

- Zenbaki osokooen multzo finituak
- Eragiketak:
  - Osoko bat gehitu multzo batean
  - Osoko bat kendu multzo batetik
  - Osoko bat multzo bat dagoen erabaki
  - Multzo bat beste batekin bildu
  - Multzo batek beste baten osoko berdinkak dituen ala ez erabaki
  - Esan ea multzo bat hutsa den
  - Multzo bateko osoko-kopurua eman
  - Multzo bateko osokooen balioien irudi testuala eman

9

## Interfazea: osokooen multzoa

```
public interface IntSet {
 public void add (int x);
 public int remove (int x);
 public boolean contains (int x);
 public IntSet union (IntSet set);
 public boolean equals (IntSet set);
 public boolean isEmpty ();
 public int size ();
 public String toString ();
}
```

L

H

DESKRIBAPENA

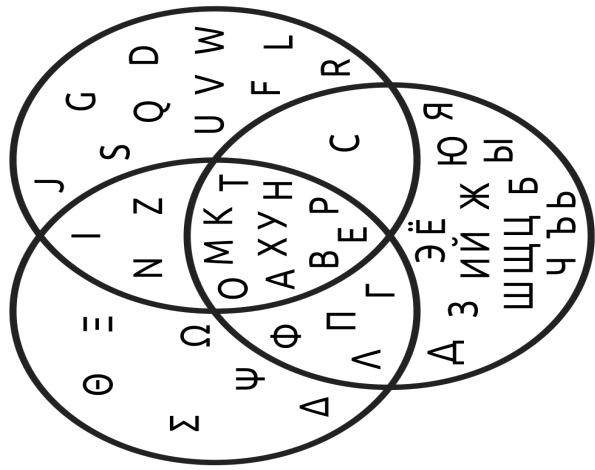
FALTA DA

(Zehaztapena)

10

Kontzeptua

- Elementuak eta barnekotasuna
  - Ez dago ordenarik
  - Ez dago errepikapenik
  - Multzoen arteko eragiketa ugari



09/03/2

Datu-Egiturak eta Algoritmoak

11

IntSet DMAaren implementazioa

```
public class MultzoB implements IntSet {
 private int[] multzoa;
 private int kardinala;

 public MultzoB() {
 multzoa = new int[100];
 kardinala = 0;
 }
 public void add (int x) {
 multzoa[kardinala] = x;
 kardinala++;
 }
 public boolean isEmpty () {
 return (kardinala == 0);
 }
...ETA HORRELA BESTE METODO GUZT
```

EDA

## IntSet IMPLEMENTATZEN DUTEN BESTE KLASSE BATZUK ERE DEFINITU DITZAKEGU

12

## Eta liburuen multzoak nahi baldin baditugu?

- IntSet interfaizeak ez digu laguntzen, nahiz eta behar ditugun eragiketak berdinak diren
- Possiblea litzateke <gauza>-ren multzoak definitzea?
  - Geroago, esango genuke zein den <gauza> objektuen klasea
- Honek **klase generikoetara** darama
- Honek **softwarearen berrerabilgarritasuna** handitzen du

13

## Klase generikoak

- Klase bat **generikoa** izango da mota-aldagai bat edo gehiago definitzen baditu

```
public interface Set<T> {
 public void add (T x);
 public T remove (T x);
 public Set<T> union (Set<T> set);
 public boolean contains (T x);
 public boolean equals (Set<T> set);
 public boolean isEmpty ();
 public int size ();
 public String toString ();
}
```

14

# Generikotasuna

- T mota generiko baten bidez parametrizatutako klaseak definitzeko aukera ematen du
- Klase parametrizatuaren egitura eta metodoek T mota generikoaren abstrakzioa egiten dute
  - Mota generikoa ez dago definituta konpilazio-denboran, baina bai egon beharko du **exekuzio-denboran**

15

## Multzoaren implementazio generikoa

```
public class ArraySet<T> implements Set<T> {
 private T[] multzoa;
 private int elemKop;

 public ArraySet() {
 multzoa = (T[]) new Object[100];
 elemKop = 0;
 }

 public void add (T x) {
 multzoa [elemKop] = x;
 elemKop++;
 }

 public boolean isEmpty () {
 return (elemKop == 0);
 }

 ... ETA HORRELA BESTE METODO GUZTEKIN
}
```

16

## Objektuen erazagupena eta sorkuntza klase generikoekin

### ■ Erazagupena:

```
Nodo<Integer> nodoI;
Nodo<Data> nodoD;
```

### ■ Sorkuntza:

```
nodoI = new Nodo<Integer>(15);
Data f = new Data(10, 11);
nodoF = new Nodo<Data>(f);
```

```
public class Nodo <T> {
 T dato;
 Nodo<T> next;

 public Nodo(T dd)
 {dato = dd;}
}
```

## 2.0.4. Iteradoreak

Creative Commons lizenziapean publikatua



### Iteradoreak?

- Zein da arazoa?
  - Bilduma bateko elementuen korritzea egiteko mekanismoa
- Nola egin?
  - Bildumako elementuei erreferentzia egindo dien objektua
  - Java-z `Iterator<T>` interfazea dago
    - `Iterator<T>` implementatzen duen klase bakoitzak objektuak era batean korritzeko diseinatuta dago

## Iterator Interfazea

- Java-ko klase estandarren liburutegian (`java.util`) diseinatuta dago: bilduma batean objektu batetik bestera mugitzeko mekanismoa eskaintzen du
  - // prozesatuko den hurrengo objektua bueltatzen du
  - public Object **next()**  
`/* Iteradorearen implementatzaren duen klasearen diseinatzaleak erabaki beharko du nola eskainiko den hurrengo objektua. Hau da, bilduma korritzeko modua erabaki behar du */`
  - // true bueltatuko du prozesatzeko elementu gehiago baleude
  - public boolean **hasNext()**
  - // iteradoreak bueltatuko duen azken elementua kenduko du
  - // bildumatik (eragiketa hau aukerazkoa da)
  - public void **remove()**

3

## Iterator Interfazea

- 
- EDA
- ```
public interface Iterator<T> {
    // prozesatuko den hurrengo objektua bueltatzen du
    public Object next()
    /* Iteradorearen implementatzaren duen klasearen diseinatzaleak erabaki beharko du nola eskainiko den hurrengo objektua. Hau da, bilduma korritzeko modua erabaki behar du */

    // true bueltatuko du prozesatzeko elementu gehiago baleude
    public boolean hasNext()

    // iteradoreak bueltatuko duen azken elementua kenduko du
    // bildumatik (eragiketa hau aukerazkoa da)

    public void remove()
}
```

Nola erabiltzen da iteradorea?

```
// demagun nireSet ArraySet<Person> objektu bat dela
// (pertsonen multzoa)

Iterator<Person> erakusleNireSet = nireSet.iterator();
Person p;
while (erakusleNireSet.hasNext()){
    p = erakusleNireSet.next();
    System.out.println(p.toString());
}
```

5

Nola erabiltzen da iteradorea?

```
public int bikoteKopurua (LinkedSet<Integer> nireMultzoa)
{
    Iterator<Integer> iteradore = nireMultzoa.iterator();
    int bikKop = 0;
    while (iteradore.hasNext() ) {
        if (iteradore.next() % 2 == 0 )
            bikKop++;
    }
    return bikKop;
}
```

6

iterator() metodoa implementatu behar da

Non agertuko da iteradore metodo bat?

- Normala da iterator() metodo bat agertzea bere elementu guztiek aztertu behar den klase bakoitzean
- Iterator() metodoaren implementazioak objektu iteradorea bueltatuko du, DMAko objektuen euskarria den egitura korritu ahal izateko
 - Array-ekin implementatutako bildumetan:


```
public Iterator<T> iterator() {
    return new ArrayIterator<T> (taula, luzera);
}
```
 - LinearNode<T> adabegiekin implementatutako bildumetan:


```
public Iterator<T> iterator() {
    return new LinkedIterator<T> (contents, count);
}
```
- **ArrayIterator<T> eta LinkedIterator<T> klaseak implementatatu behar dira**

7

Array-en iteradorea

```
import java.util.*;

public class ArrayIterator<T> implements Iterator<T> {
    private int count; // bildumako elementu-kopurua
    private int current; // korritzearen uneko posizioa
    private T[] items;

    // Emandako bilduma korritzeko iteradorea hasieratzen du
    public ArrayIterator (T[] collection, int size) {
        items = collection;
        count = size;
        current = 0;
    }

    // true bueltatuko du bildumak elementu bat gehiago baldin badu
    // (raindik ez izan bisitatu)
    public boolean hasNext(){
        return (current < count);
    }

    // Bildumako hurrengo elementua bueltatzen du, korritzearen arabera
    public T next(){
        current++;
        return items[current-1];
    }
}
```

8

LinearNode<T> nodoetarako iteradorea zerrenda estekatuetarako (Lewis & Chase 2010)

EDA

```

public class LinkedIterator<T> implements Iterator<T> {
    //private int count; // liburuan agertzen da baina ez da beharrezko
    private LinearNode<T> current; // uneko posizioa

    // Emandako bilduma korritzeko iteradorea hasieratzen du
    public LinkedIterator(LinearNode<T> collection, int size) {
        current = collection;
        //count = size; Ez da erabilten. Liburua jarraitzearen utzi dugu
    }

    // true bueltatuko du bildumak elementu bat gehiago baldin badu, oraindik bisitatugabea
    public boolean hasNext() {
        return (current != null);
    }

    // Bildumako hurrengo elementua bueltatuko du, korritzearen arabera
    public T next() {
        if (!hasNext())
            throw new NoSuchElementException();
        T result = current.getNext();
        current = current.getNext();
        return result;
    }

    // Ezabaketarako eragiketa ez dago implementatuta
    public void remove() throws UnsupportedOperationException {
        throw new UnsupportedOperationException();
    }
}

```

9

Matrizeak aztertzeko iteradorearen adibidea

```

import java.util.Iterator;

public class Matrize<T> {
    private T[][] m;
    private static int N = 20;
    private static int M = 50;

    public Matrize() {
        m = (T[][]) new Object[N][M];
    }

    public Iterator<T> NireIteradorea() {
        return new MatrizeIterator<T>(m, N, M);
    }
}

```

EDA

10

Matrizeak aztertzeko iteradorearen adibidea

```

import java.util.Iterator;
public class MatrizeIteradore<T> implements Iterator<T> {

    T[][] itMat;
    int i, j;
    int lerroMax, zutMax;

    public MatrizeIteradore(T[] [] mat, int zutKop, int lerroKop) {
        itMat = mat;
        lerroMax = lerroKop;
        zutMax = zutKop;
        i = 0;
        j = 0;
    }

    public boolean hasNext() {
        if ((i < lerroMax) && (j < zutMax)) return true;
        else return false;
    }

    public T next() {
        T temp = itMat[i][j];
        j++;
        if (j >= zutMax) {
            i++;
            j = 0;
        }
        return temp;
    }

    public void remove() {} // egin gabe!!!!
}

```

11

Beste aukera bat: klase pribatu bat iteradorea emateko

```

public class nireArrayList<T> {

    private T[] taula;
    private final int OINARRIZKO_TAMAINA = 100;
    private int lehenLibrea;

    public nireArrayList() { // eraikitzailea...}

    public Iterator<T> it() { return new It(); }

    private class It implements Iterator<T>{
        int i = 0; // Hasieraketa

        public T next(){
            if (i < lehenLibrea) { i++; return taula[i - 1]; }
            else return null;
        }

        public boolean hasNext(){
            if (i < lehenLibrea) { return true; } else return false;
        }

        public void remove() {} // implementatu barik
    }
}

```

Estructuras enlazadas

12

Beste aukera bat: klase pribatu bat iteradorea emateko

```
public class nireList<T> {

    private Link<T> first;

    public Iterator<T> iterator() { return new ListIterator(); }

    private class ListIterator implements Iterator<T> {

        private Link<T> current = first; // Hasiera keta

        public boolean hasNext() { return current != null; }

        public void remove() { throw new UnsupportedOperationException(); }

        public T next() {
            if (!hasNext()) throw new NoSuchElementException();
            T item = current.data;
            current = current.next;
            return item;
        }
    } // private class
}
```

Estructuras enlazadas

13

Foreach moduko iterazioak

- Iteragariak diren klaseetan, for sententzia beste modu honetara ere erabil daiteke:

```
public static void imprimatu(ArrayMultzoa<Integer> multzoa)
{
    System.out.print("[ ");
    for (int elementua : multzoa) {
        System.out.print(elementua + " ");
    }
    System.out.println("]");
}
```

Datu egiturak



Pilak eta ilarak

1



Aurkibidea



•Pilak

- Zer dira?
- Ezagutza
- Metodo nagusiak
- Adibideak

•Ilarak

- Zer dira?
 - Ezagutza
 - Metodo nagusiak
 - Adibideak
- Implementazio adibideak
 - Array-etan oinarritura
 - Zerrendetan oinarritura

2





Pilak



3

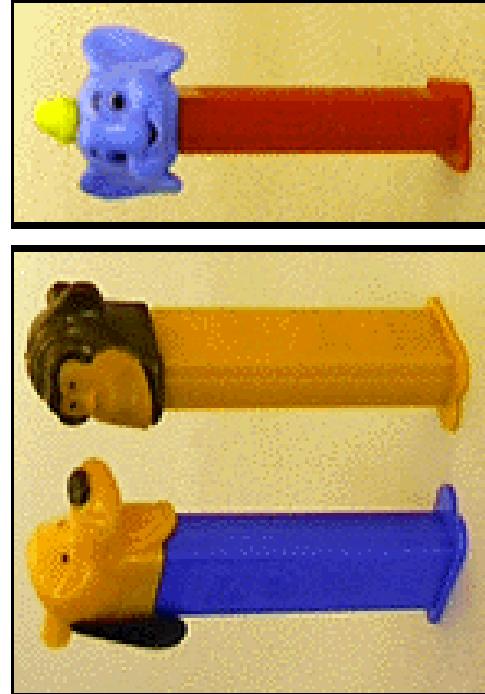
Pilak Zer dira?. Ezaggarriak



- Datu egitura **lineala** non:
 - Txertatu den elementu berriena atzitu daitake soilik (**top** edo gailur)
 - Objektuak **mutur batetik** txertatzen eta ezabatzen dira **LIFO** (**Last In First Out**) irizpidea jarraituz.
- Egokiak dira:
 - **Azken** elementua atzitu behar dugunean soilik.
 - Zerrenda bat **alderantzikatu** nahi dugunean.

4

Pilak Adibidea

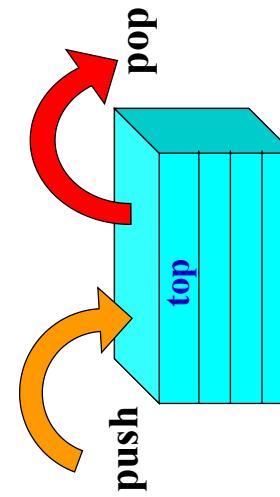


5



Pilak Metodo nagusiak

- Mutur batetik txertatu (pilaratu): **push (x)**
- Mutur berdinietik atera (despilatu): **pop ()**



Eskuragarri dagoen elementu bakarra azken txertatua da (gailur): **top**

6



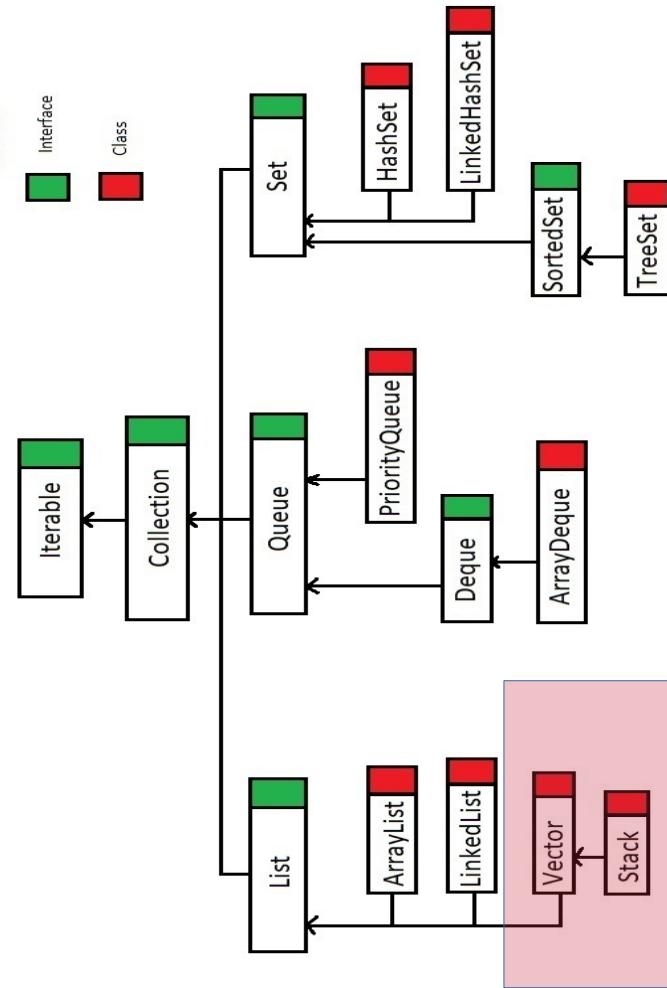
Pilak Metodo nagusiak eta metodo laguntzaileak

Metodo nagusiak	Esanahia
void push (T elem)	Pilaren gailurrean elementu bat txertatzen da
T pop()	Pilaren gailurrean dagoen elementua ezabatzen da posizio horretan zegoen elementua izuliz

Metodo laguntzaileak	Esanahia
int size()	Pilaren tamaina itzultzen du
boolean isEmpty()	True pila hutsa bada eta false kontrako kasuan
T peek()	Gailurrean dagoen elementua itzultzen du gailurretik ezabatu gabe.

7

HIERARCHY OF JAVA COLLECTION FRAMEWORK





Pilak Adibideak

- Pilak konpiladoreetan:
 - **Simboloen oreka** konprobatzeko
 - **Metodoen deiak gordetze**: Errekurtsoa
 - Dei bakoitzean gordetzen da:
 - Metodoen aldagai lokalak (kontextu)
 - Deia egin den lerroa

9



Pilkak: metodoen deialdiak gordetze

```
1 public class Faktoriala{  
2     public static void main(String args[]) {  
3         int param = Integer.parseInt(args[0]);  
4         long emaitza = fakt(param);  
5         System.out.println( args[0] + "!" = " + emaitza);  
6     } // main amaiera  
7     public static long fakt(int n) {  
8         if(n<=1)  
9             return 1;  
10        else  
11            return n*fakt(n-1);  
12    } // fakt amaiera  
13 } // Faktoriala amaiera
```

Pilak

Adibide 1: Parentesi konprobaketa



- Ondo:
 - 
 - ()
 - ((()))
- Simboloen oreka konprobatzeko erregelak:
 - Oinarrizkoak: ()
 - Sekuentzia: ()()
 - Habiratuak: (())

11

Pilak

Adibide 1: Parentesi konprobaketa



• Erregelak:

- (aurkitzen dugun bakoitzean, pilan txertatzen dugu.
-) aurkitzen dugun bakoitzean, gailurrean dagoen (ateratzen dugu
- Parentesi katea **zuzena** da, kate guztia korritu dugunean **pila hutsik badago**.

12

Pilak

Adibide: Parentesi konprobaketa $((())())()$



Erregelak:

Parentesi irekia: pilaratu
Parentesi itxia: despilatu

$((())())())()$

13

Pilak

Adibide 1: Parentesi konprobaketa $((())())()$



Zuzena: katea korritu dugu
eta **pila hutsik dago**



14

Labirintoa zeharkatzen

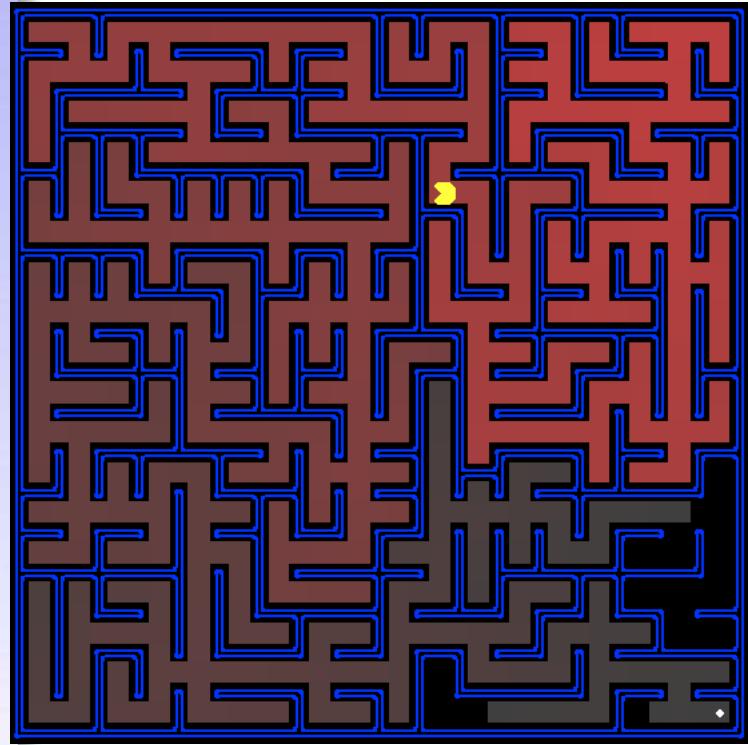


```
private int [ ] [ ] labirinto =  
{ {1,1,1,0,0,1,0},  
{1,1,0,1,0,1,0},  
{0,1,0,1,1,1,1},  
{1,1,1,0,1,0,0},  
{0,0,1,1,1,0,1},  
{1,0,0,0,1,0,1},  
{1,1,1,1,1,0,0},  
{1,0,0,0,1,1,1},  
{1,1,1,1,1,1,1} };  
  
labirinto [errenkada] [zutabe] == 1 "bidea da"  
labirinto [errenkada] [zutabe] == 0 "horma da"
```

15



Labirintoa zeharkatzen



16



Labirintoa zeharkatzen

```
public boolean traverse () {  
    Poner posición inicial en la cima de la pila  
    Marcar esa posición como visitada  
    while "la posición no sea la salida" Y  
        "tenga esperanza de poder llegar a la salida"  
        {  
            posición = extraer la cima de la pila  
            if "la posición es la salida" terminar el while  
            else {  
                para cada casilla posible de avance desde posición:  
                    apilar esa casilla /* puede verse como  
                    una tarea que queda por hacer: intentar  
                    el camino desde esa casilla*/  
                    Marcar esa posición como visitada  
                }  
            } //end while  
            if "la posición es la salida" return true  
            else return false  
        }  
}
```

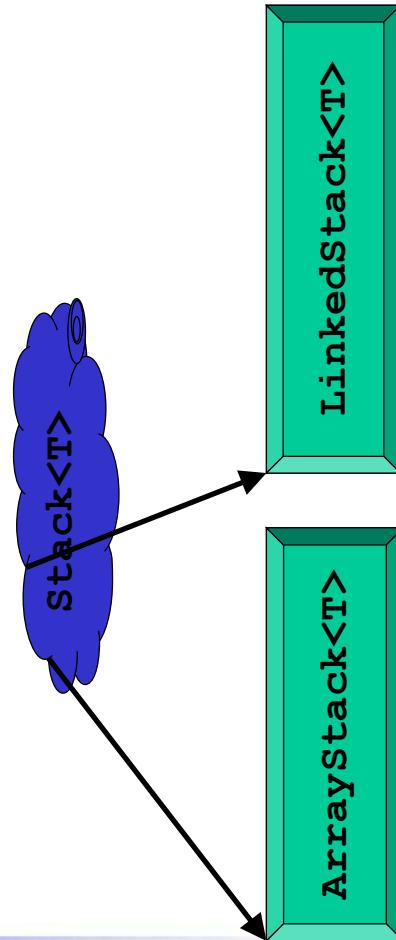


Pila interfazea

```
public interface Stack<T> {  
    public int size();  
    public boolean isEmpty();  
    public void push(T o);  
    public T pop();  
    public T peek();  
}
```

Pilkak:

Implementazio desberdinak interfaze batentzat



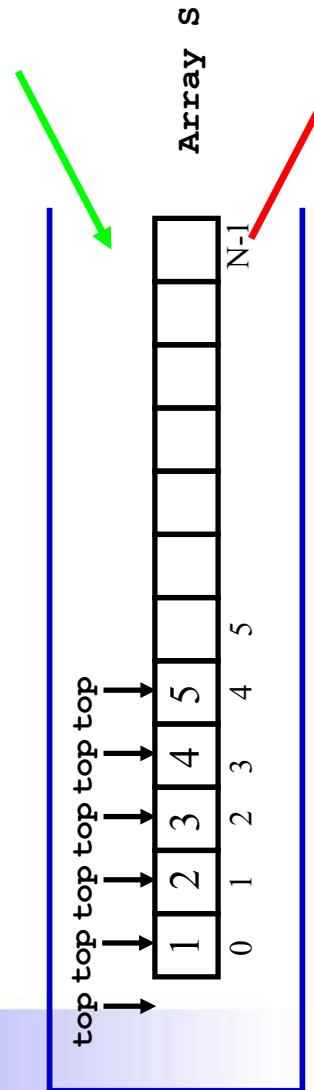
19

Pilkak:

Array-eten oinarritutako implementazioa



S array baten bidez implementatutako **pila** batean o objektu bat txertatu nahi dugu



20

```

top = top + 1; //top etiketa posizio bat mugitu
s[top] = o; //top posizio berrian elementua txertatzen da
  
```



Pilak:

Array-etan oinarrituako implementazioa

```
public class ArrayStack<T> implements Stack<T> {  
    public static final int CAP = 1000;  
    private int maxSize; // size of stack array  
    private T[] stackArray;  
    private int top; // top of stack  
  
    public ArrayStack() {this(CAP);} // constructor  
    public ArrayStack(int alturaMax) {  
        maxSize = alturaMax; // set array size  
        stackArray = (T[]) (new Object[maxSize]);  
        // create array  
        top = -1; // no items yet  
    }  
}
```

21



Pilak:

Array-etan oinarrituako implementazioa

```
public void push(T elemento) {  
    // put item on top of stack  
    stackArray[++top] = elemento;  
    // increment top, insert item  
}  
  
public T pop() {  
    // take item from top of stack  
    return stackArray[top--];  
    // access item, decrement top  
}
```

22



Pilak:

Array-etan oinarritutako implementazioa

```
public T peek() { // peek at top of stack  
    return stackArray[top];  
}  
  
public boolean isEmpty() { // true if empty  
    return (top == -1);  
}
```

```
23  
  
public int size() {  
    return (1+top);  
}  
  
}
```



Pilak: iteradore bat nahi dugu?

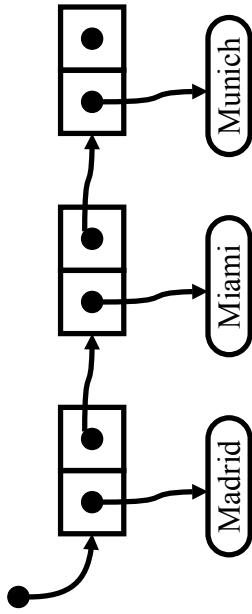
```
public Iterator<T> iterator() {  
    return new ReverseArrayIterator(); }  
  
Nested class  
  
public class ReverseArrayIterator implements Iterator<T> {  
  
    private int i = top;  
  
    public boolean hasNext() {  
        return i >= 0; }  
  
    public Item next() {  
        return stackArray[i--]; }  
  
    public void remove() {  
        throw new UnsupportedOperationException(); }  
    } // ReverseArrayIterator  
} // ArrayStack<T>
```

24



Pilak:

Zerrenda esteekatueta oinarritutako implementazioa



25



Pilak:

Zerrenda esteekatueta oinarritutako implementazioa

```
public class LinkedStack<T> implements Stack<T>{  
    private LinearNode<T> top;  
    private int count;  
    public LinkedStack() { // eraikitzailea  
        top = null;  
        count = 0;  
    }  
    public boolean isEmpty() {  
        return (top == null);  
    }  
    public int size() { return count; }  
}
```

26

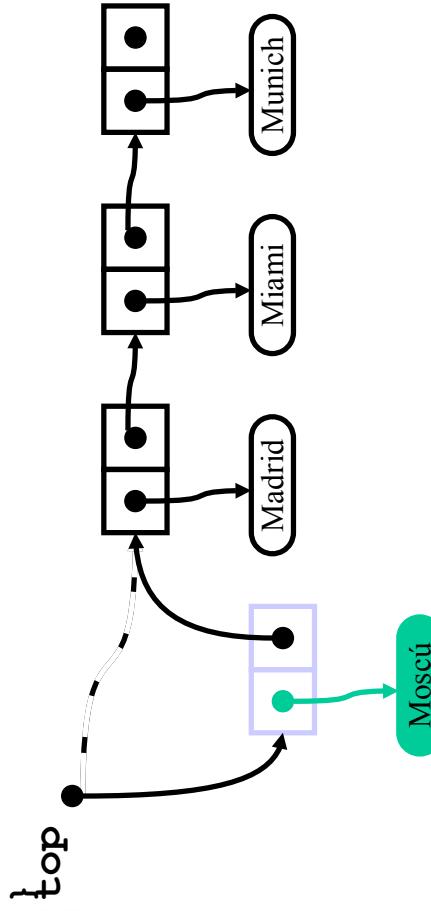
Pilak:



Zerrenda estekatueta oinarritutako implementazioa
txertaketa: **push()**

```
public void push(T e) {
```

```
    top = new Node(e);
```



27

Pilak:



Zerrenda estekatueta oinarritutako implementazioa atzipena:
peek()

```
public T peek () {  
    // Precondition: not empty
```

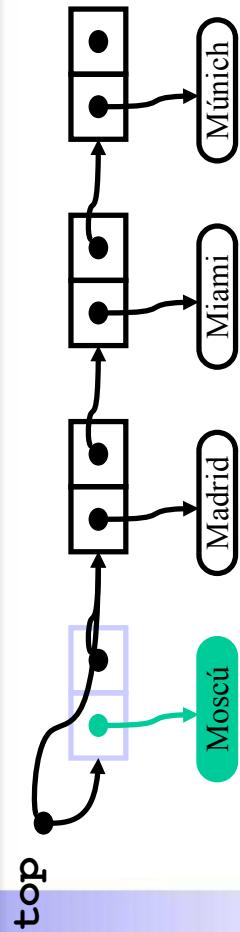
```
}
```

28

Pilak:



Zerrenda estekatueta oinarritutako implementazioa
ezabaketa: **pop()**



```
public T pop() {  
    // Precondition: not empty  
    } // LinkedStack
```

29

Datu egitura llarak



30



Ilarak

Zer dira? Ezaggarriak

- Datu egitura **lineala** non:
 - Atzipena, hasieran txertatutako elementuari murriztua dago (**elementu zaharrena**)
 - Elementuak mutur batetik txertatzen dira eta beste muturretik ateratzen dira **FIFO** (**F**irst **I**n **F**irst **O**ut) irizpidea jarraituz.

31



Ilarak: Adibideak

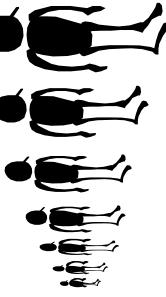
- **Autobuseko ilara**

- Lehenengoa sartzen da, lehenengoa etorri dena.

- Ilaran denbora gehiago daramana

- **Inprimaketa-ilara**

- Azken bidalitako fitxategia, azkeneko ateratzen dena izango da.



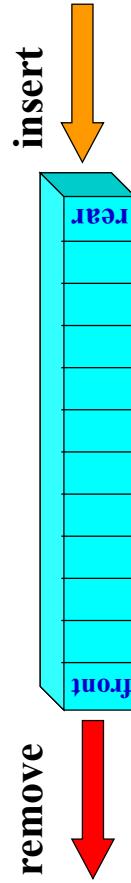
32



llarak

Metodo nagusiak

- Mutur batetik txertatu (ilaratu): `insert(x)`
- Beste muturretik atera (desilaratu): `remove()`



Eskuragarri dago, llararen **lehen** elementua (**front**).

33



llarak

Metodo nagusiak eta metodo laguntzaileak

Metodo nagusiak	Esanahia
<code>void insert(T elem)</code>	Elementu bat txertatzen du llararen bukaeran.
<code>T remove()</code>	llararen hasierako elementua ezabatzentzu eta bere edukia itzultzen du . Hutsa bada, null bueltatuko du.

Metodo laguntzaileak	Esanahia
<code>int size()</code>	llararen tamaina itzultzen du
<code>boolean isEmpty()</code>	true, llara hutsa bada eta false kontrako kasuan
<code>T front()</code>	llararen hasieran dagoen elementua itzultzen du llaratik atera gabe

34

Mezu baten kodeketa

- Mezuko letra bakoitzak aldatuko da, bere ordez, letra horri “dagokion” gakoak eman dako alfabetoko distantzian kokatuta dagoen letra jarriaz
 - **Gakoa** zenbakien sekuentzia bat da
 - Adibidez, gakoa {3, 15, -4, 7, -9, 5} bada, orduan lehenengo letra 3ko distantzian dagoen letragatik aldatuko da, bigarrena 15eko distantzian eta horrela letra guztiekin. 5a erabili eta gero, lehenengo gaiarekin jarraituko da, mezu oso kodetu arte
 -
 - Ikus 5.2 atala [Lewis, Chase] liburuan

Ilarak

35

Erroaren bidezko ordenazioa (radix sort)

- Zenbakien sekuentzia bat ordenatu behar da (digitu-kopuru maximoa ezagututa)
 - Ikus *radix sort* Edozein liburutan (wikipedia)
 - Ikus 7.4 atala [Lewis, Chase] liburuan

Ilarak

36

Erroaren bidezko ordenazioa. Adibidea

- Gehienez 3 digituko zenbakи hamartarrak
- Sekuentzia ez-ordenatua:
 - 170, 45, 75, 90, 2, 24, 802, 66.
 - Edo beste era batera:
 - 170, 045, 075, 090, 002, 024, 802, 066

Iharak

37

Lehen fasea

- 170, 045, 075, 090, 002, 024, 802, 066
 - Ilaretan sartuko ditugu **pisu gutxieneko digituagatik** (num%10)
 - Berriro bilduko ditugu ilara batean, 0 ilaratik hasita, 9 ilarara iritsi arte:
 - 170, 090, 002, 802, 024, 045, 075, 066
- | | |
|---|---------------------------|
| 0 | 17 <u>0</u> , 09 <u>0</u> |
| 1 | |
| 2 | 00 <u>2</u> , 80 <u>2</u> |
| 3 | |
| 4 | 02 <u>4</u> |
| 5 | 04 <u>5</u> , 07 <u>5</u> |
| 6 | 06 <u>6</u> |
| 7 | |
| 8 | |
| 9 | |

Iharak

38

Bigarren fasea

- 170, 090, 002, 802, 024, 045, 075, 066
Ilaretan sartuko ditugu **pisu gutxieneko bigarren digituagatik**
((num/10)%10)

0	0 <u>0</u> 2, 8 <u>0</u> 2
1	
2	0 <u>2</u> 4
3	
4	0 <u>4</u> 5
5	
6	0 <u>6</u> 6
7	1 <u>7</u> 0, 0 <u>7</u> 5
8	
9	0 <u>9</u> 0

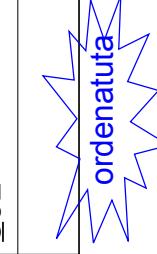
Berriro bilduko ditugu ilara batean, 0 ilaratik hasita, 9 ilarara iritsi arte:

- 002, 802, 024, 045, 066, 170, 075, 090

Ilarak

39

Hirugarren fasea (eta azkena)

- 002, 802, 024, 045, 066, 170, 075, 090
Ilaretan sartuko ditugu **pisu gutxieneko hirugarren digituagatik**
((num/100)%10)
- Berriro bilduko ditugu ilara batean, 0 ilaratik hasita, 9 ilarara iritsi arte:
002, 024, 045, 066, 075, 090, 170, 802 

Ilarak

40

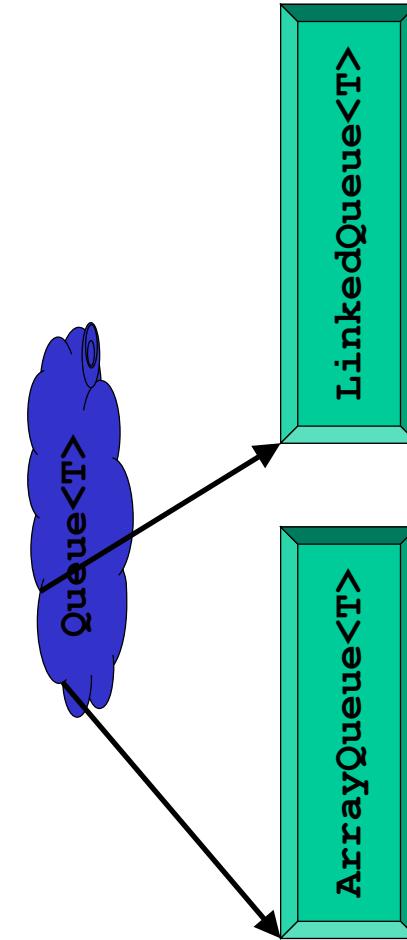
||arak ||araren interfazea



```
public interface Queue<T> {
    public int size();
    public boolean isEmpty();
    public void insert(T o);
    public T remove();
    public T front();
}
```

41

||arak: Implementazio desberdinak interfaze batentzat



42

||arak:

Zerrenda esteekatueta oinarritutako implementazioa



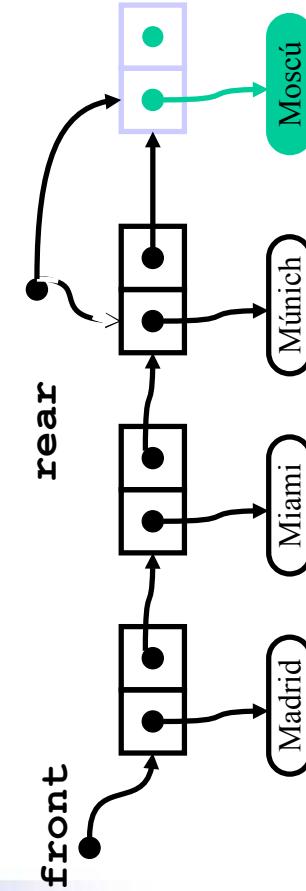
```
public class LinkedQueue<T> implements Queue<T> {
    private LinearNode<T> front;
    private LinearNode<T> rear;
    private int count;
    public LinkedQueue() {
        front = null;
        rear = null;
        count = 0;
    }
    public boolean isEmpty() { return front==null; }
    public int size() { return count; }
}
```

43

||arak:

Zerrenda esteekatueta oinarritutako implementazioa
Txertaketa: **insert**

```
public void insert(T e) {
}
}
```



44



llarak:

Zerrenda estekatuetan oinarritutako implementazioa
Ezabaketa: `remove`

```
public T remove () {  
    // Precondition: not empty  
  
    }  
}
```

45



llarak:

Zerrenda estekatuetan oinarritutako implementazioa
Hasierako elementua: `first`

```
public T front () {  
    // Precondition: not empty  
    return (front.getElement ()) ;  
  
}
```

46

Illaia: arrayen bidezko implementazioa



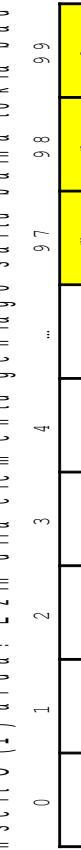
```
i.insert('e');
```



```
i.remove(); i.remove(); // 2 elementu ateratzen
```



```
insert(1) alda? Ezin dira elementu gehiago sartu baina tokia badago!
```

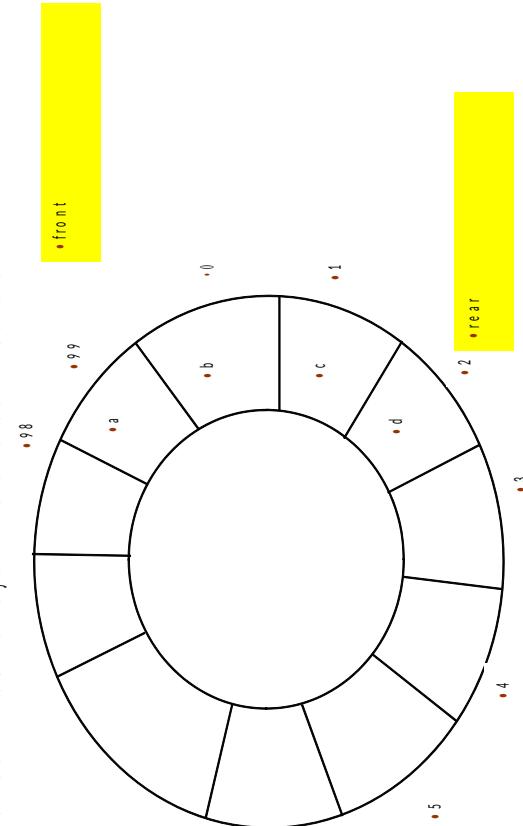


Colas

Illaia: arrayen bidezko implementazioa

- Lehen soluzioa: elementuak mugitu tokaea egiteko => insert(0(n)) da!!

- Bigarren soluzioa: "ikusi" arraya zirkularra balitz bezala



Illaia

Ilara: arrayen bidezko implementazioa



- Arazoa: zelan mugitu aurkeoa?

Hurrengoa	Elementuaren indizlea (N)
5	6
6	7
...	
98	99
99	0
0	1

49

$$\text{Hurrengoa}(N) = (N + 1) \% 100$$

EDA

Ilara: arrayen bidezko implementazio "zirkularra" (0)

				A	B	C	D		
0	1	2	3	4	5	6	7	8	9
ilara	front	rear							

nItem s = 4

$$\maxSize = 10$$

D							A	B	C
0	1	2	3	4	5	6	7	8	9
ilarak	front	rear							

Ilarak



Ilarra: arrayen bidezko implementazio “zirkularra” (1)

```
public class ArrayQueue<T> implements Queue<T> {
    private int maxSize;
    private T[] taula;
    private int front;
    private int rear;
    private int nItems;

    public ArrayQueue(int s) {
        maxSize = s;
        taula = (T[]) new Object[maxSize];
        front = 0;
        rear = -1;
        nItems = 0;
    }

    public void insert(T j) {
        if(rear == maxSize-1)
            rear = -1;
        taula[++rear] = j;
        nItems++;
    }
}
```

51



Ilarra: arrayen bidezko implementazio “zirkularra” (2)

```
public T remove() {
    T temp = taula[front++];
    if(front == maxSize)
        front = 0;
    nItems--;
    return temp;
}

public T front() { // Precondition: not empty
    return taula[front];
}

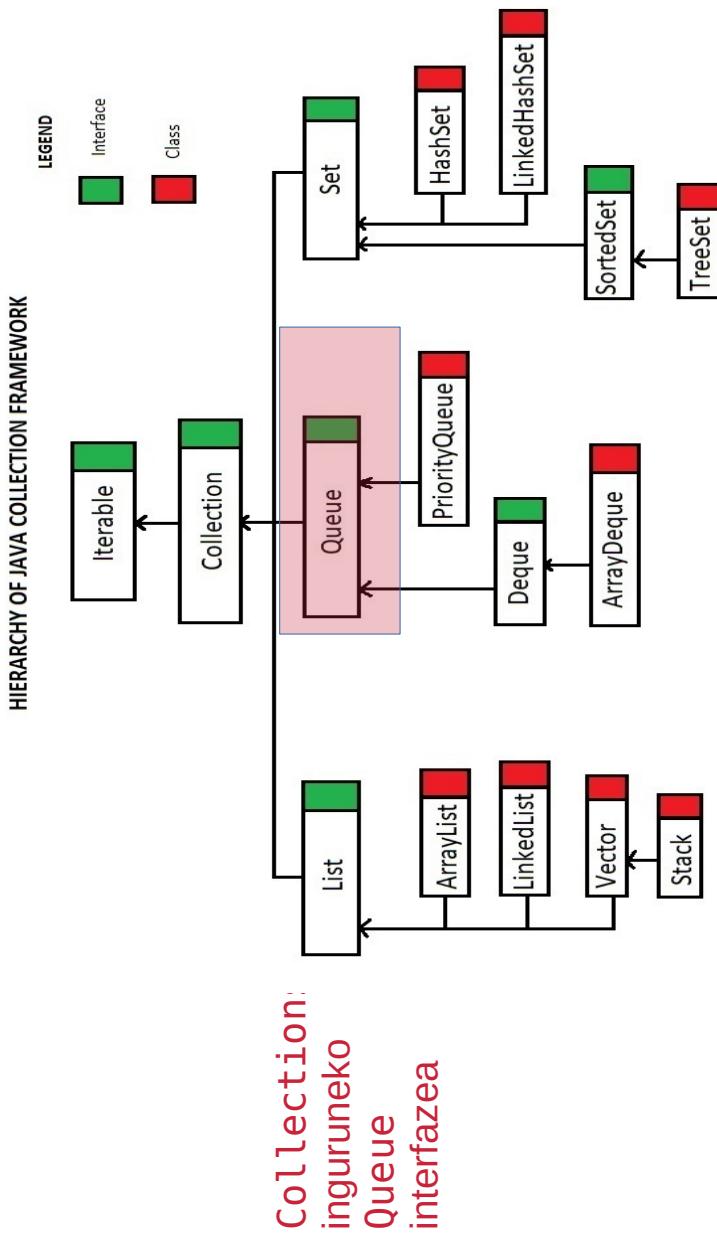
public boolean isEmpty() { return (nItems==0); }

public boolean isFull() { return (nItems==maxSize); }

public int size() { return nItems; }

} // ArrayQueue
```

52



03/09/2021

Datu-Egiturak eta Algoritmoak

53

Queue interfazea

- Collections bilduman, Queue interfazea da
 - Beraz, klase batean implementatu behar da erabili aurretik
 - Nola implementatu Queue?

03/09/2021

Datu-Egiturak eta Algoritmoak

54

Bi hurbilpen

- `ArrayQueue<T>`
- `LinkedListQueue<T>`

03/09/2021

Datu-Egiturak eta Algoritmoak

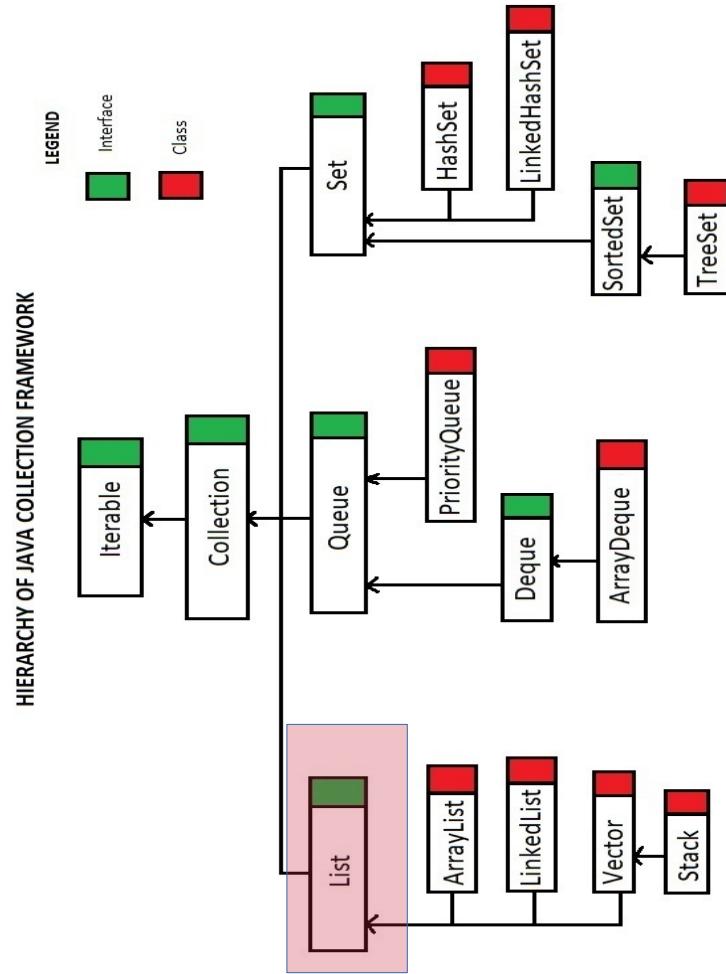
55



Interfaze bat Implementazio desberdin asko

Egitura Zer? Nola? Implementazioa	Pila	Ilara	Array	Vector	LinkedList	DoubleLinkedList	Beste egitura linealak

Collections inguruneko List interface



Listak

DEA

Creative Commons lizenziapean publikatua

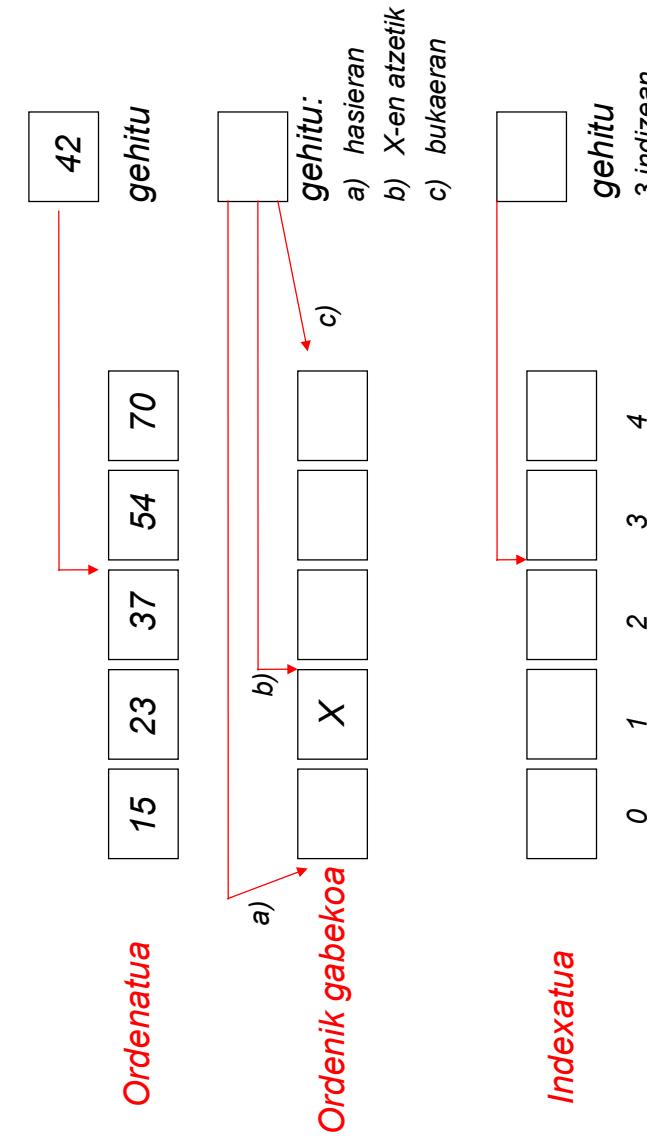


Lista mota desberdinak

- *Lista edo zerrenda bat elementuen bilduma linealetzat da. Beraien ezaugarrrien arabera, 3 mota nagusitan sailkatuko ditugu:*
 - ***Lista ordenatuak***
 - Elementuak ordena-erlazio baten arabera daude ordenatuta
 - ***Lista ez-ordenatuak***
 - Elementuak ez daude ezaugarrien arabera ordenatuta, sartu diren ordenarekiko baizik
 - ***Lista indexatuak***
 - Elementuak zenbakizko indize baten bidez atzitu daitezke

3

Lista mota desberdinak



4

Lista mota guztiek dauzkaten eragiketak

Eragiketa	Deskribapena
<i>T removeFirst()</i>	<i>Listako lehen elementua ezabatzen du</i>
<i>T removeLast()</i>	<i>Listako azken elementua ezabatzen du</i>
<i>T remove(T elem)</i>	<i>Listako elementu jakin bat ezabatzen du</i>
<i>T first()</i>	<i>Listako lehen elementuaren atzipena</i>
<i>T last()</i>	<i>Listako azken elementuaren atzipena</i>
<i>boolean contains(T elem)</i>	<i>Listak ea elementu jakin bat duen ala ez</i>
<i>boolean isEmpty()</i>	<i>Ea lista hutsa den</i>
<i>int size()</i>	<i>Listaren elementu-kopurua ematen du</i>

5

Lista ordenatuek bakarrik daukaten eragiketa

Eragiketa	Deskribapena
<i>void add(T elem)</i>	<i>Elementu berria gehituko du listan (ordenatura dagokion tokian)</i>

6

Lista ez-ordenatuek bakarrik daukaten eragiketak

Eragiketa	Deskribapena
<code>void addToFront(T elem)</code>	Elementua gehituko du listaaren hasieran
<code>void addToRear(T elem)</code>	Elementua gehituko du listaaren bukaeran
<code>void addAfter(T elem, T target)</code>	elem elementua gehituko du beste elementu jakin baten atzean (target), elementu hau listan badagoelarik

7

Lista indexatuen eragiketa bereziak

Eragiketa	Deskribapena
<code>void add(int i, T elem)</code>	Elementu berria gehituko du adierazitako indizean
<code>void set(int i, T elem)</code>	Zehaztutako indizean dagoen elementua esleituko du
<code>T get(int i)</code>	Emandako indizearen elementuaren atzipena bueltatzen du
<code>int indexOf(T elem)</code>	Elementu horren indizea bueltatuko du
<code>T remove(int i)</code>	Emandako indizeari dagokion elementua listatik kenduko da eta bere atzipena bueltatuko da

Pentsatu nola implementatu daitekeen lista ez-ordenatua bat lista indexatua erabiliz

8

Lista indexatuaren implementazioa Array baten bidez

DEA

```
public class ListaArray<T> implements ListaIndexatuDMA<T> {
    private int maxSize; // array-aren tamaina
    private T[] lista;
    private int luzera; // libre dagoen lehen elementuaren indizea eta
    // listaren elementu-kopuruua

    public ListaArray(int maxLuzera) {
        maxSize = maxLuzera;
        lista = (T[])new Object[maxSize];
        luzera = 0;
    }

    public T first () // listako lehenengoa
    {
        return lista[0];
    }

    public T last () // listako azkena
    {
        return lista[luzera - 1];
    }
}
```

9

Lista indexatuaren implementazioa Array baten bidez

DEA

```
public T removeLast () { // listako azkena kentzen du
    T lag = lista[--luzera]; // listako azkena
    lista[luzera] = null; // erreferentzia ezabatu
    return lag;
}

public boolean isEmpty () { // Ea lista hutsik dagoen ala ez
    return (luzera == 0);
}
```

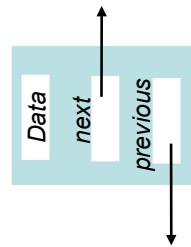
- Falta diren eragiketen implementazioa ariketa moduan uzten da.

10

Listaren implementazioa egitura estekatuaren bidez

DEA

- Aukera bat esteka bateko adabegiak erabiltzea da (`LinearNode<T>` bezalakoak). Hau ariketa moduan uzten dugu.
- Beste aukera bat bi estekako adabegiak erabiltzea da: bata hurrengoari eta bestea aurrekoari



11

`DoubleNode<T>` klasea

class DoubleNode<T> {

```

private DoubleNode<T> next;
private DoubleNode<T> previous;
private T element;

```

```

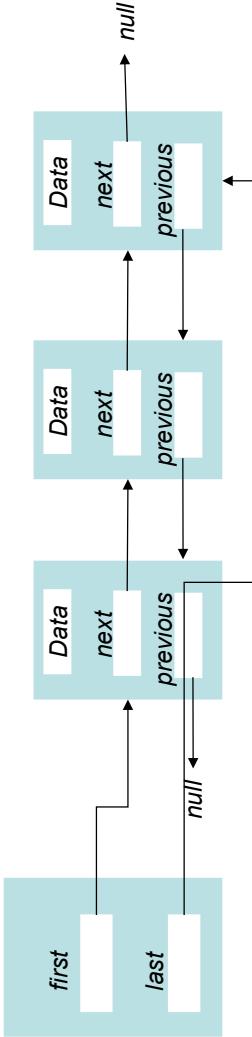
public DoubleNode() {}
public DoubleNode(T elem) {}
public DoubleNode<T> getNext() {}
public void setNext(DoubleNode<T> node) {}
public DoubleNode<T> getPrevious() {}
public void setPrevious(DoubleNode<T> node) {}
public T getElement() {}
public void setElement(T elem) {}
}

```

DEA

12

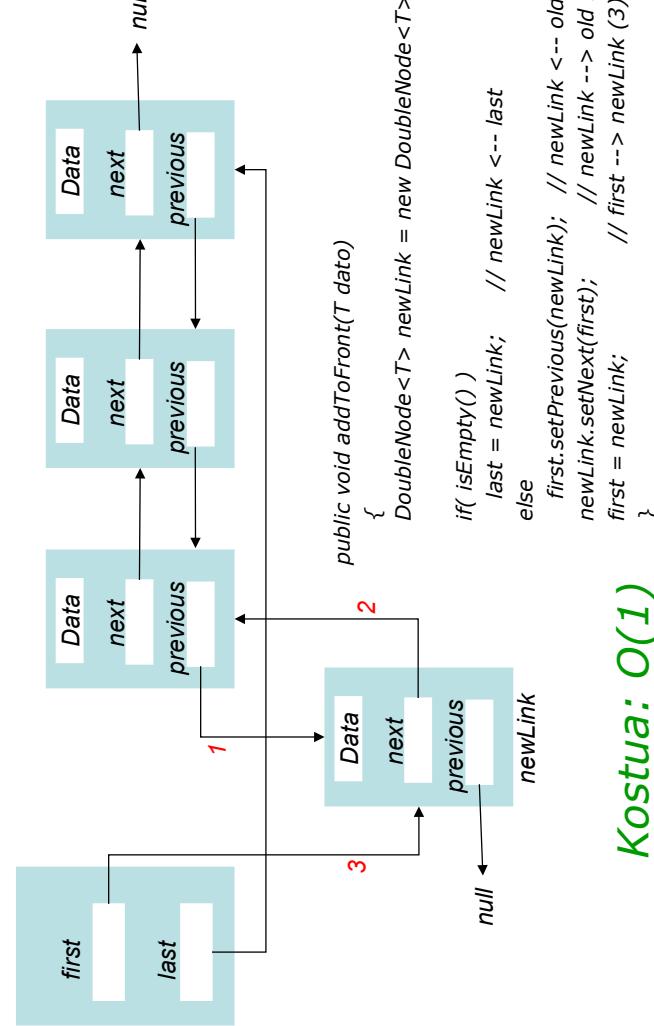
Estekadura bikoitzeko listak (0)



```
class DoubleNode<T>
{
    private T data;           // informazioa
    private DoubleNode<T> next;   // hurrengo adabegia
    private DoubleNode<T> previous; // aurreko adabegia
    .....
}
```

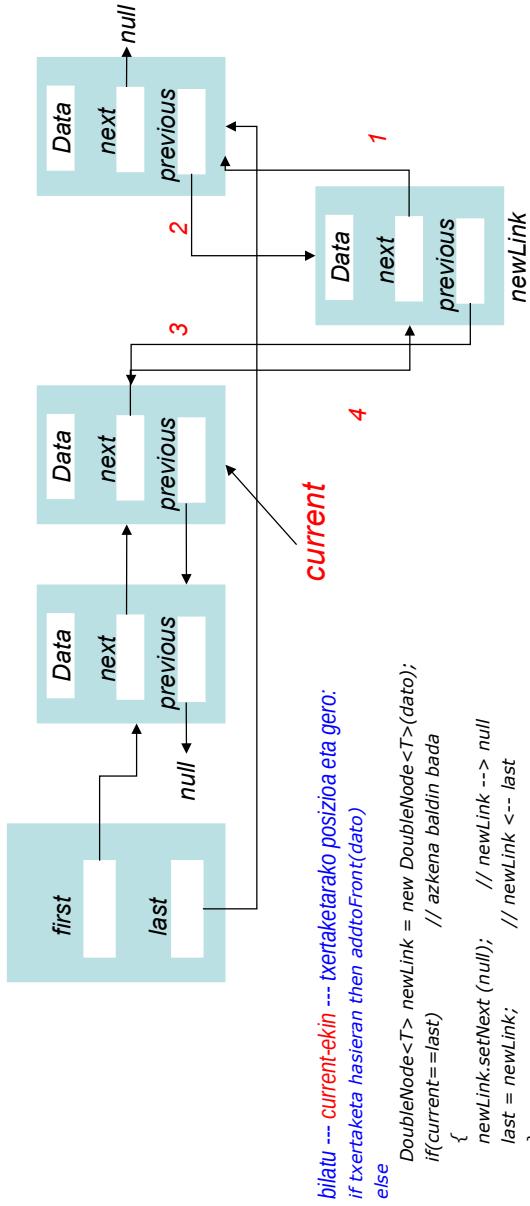
13

Hasieran txertatu (1)



14

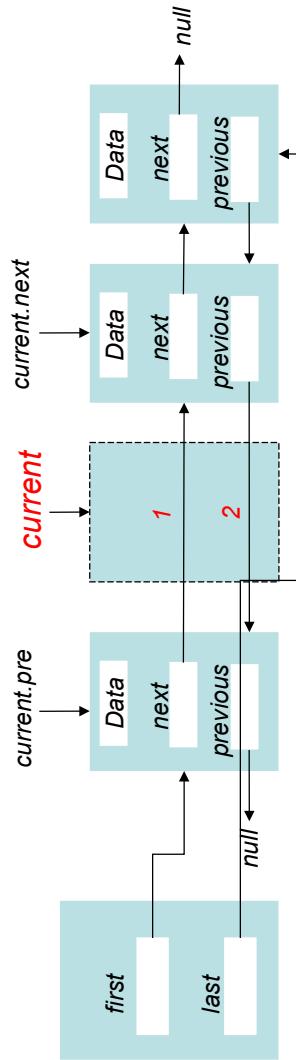
Txertaketa (2)



Kostua: $O(n)$
bilaketagatik

15

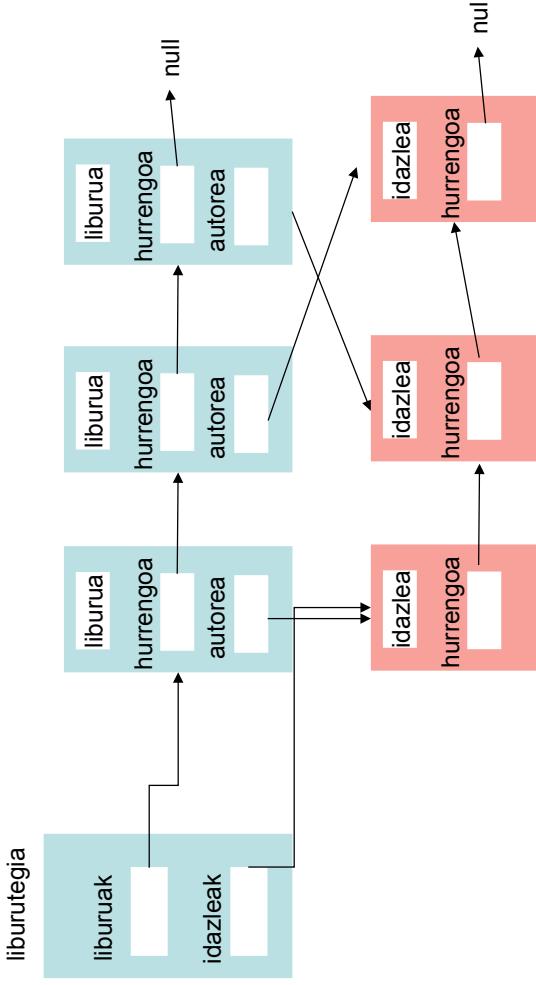
Ezabaketa (3)



Kostua: $O(n)$
bilaketagatik

16

Listetan oinarritutako egitura konplexuagoak. Adibidea: liburutegia



09/03/2021	Datu-Egiturak eta Algoritmoak	17
------------	-------------------------------	----

DEA

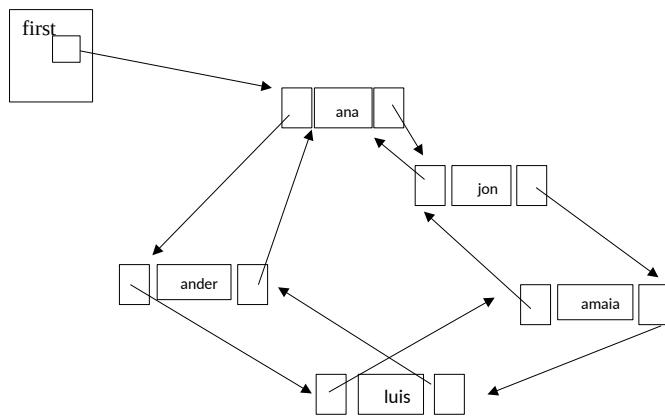
Irakurketa

- Javaren API Collections-ek dauzka lista indexatuaren implementazioak
 - *ArrayList*
 - *LinkedList*

- [Lewis, Chase 2010]

- 6. kapitulu

1. (1,5 puntu) Estekadura bikoitzeko zerrenda zirkularra dugu:



Funtzio hau implementatu nahi dugu:

```
public DoubleNode<T> {
    T data;
    DoubleNode<T> next;
    DoubleNode<T> prev;
}

public DoubleLinkedList<T> {
    DoubleNode<T> first;

    public T azkenaLortu(Integer jauzia)
        // aurre: listak gutxienez elementu bat du
        // post: elementu bakarreko lista bueltatuko du, zerrrendatik pausu
        // bakoitzean elementu bat kenduz, unezotik "jauzi" elementu pasatuz
        // (erlojuko orratzen norabidean) elementu bakarra geratu arte
}
```

Adibidez, azkenaLortu(4) deiak "jon" bueltatuko du (elementu hauek ezabatu ondoren: ander, luis, ana eta amaia).

Hau eskatzen da:

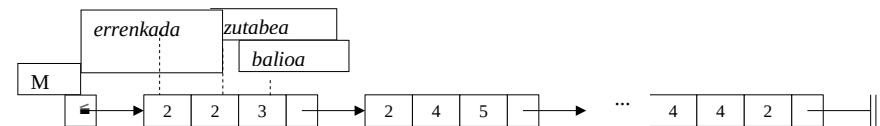
- Algoritmoa implementatu
- Kostua kalkulatu, modu arrazoituan.

1.- Matrize batuketa (2,5 puntu)

Matrize sakabanatuetaun posizio gehienetan zero balioa egoten da. Adibidez:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

Mota honetako matrizea zerrenda estekatu simplea erabiliz adierazi daiteke. Matrizean 0 balioa ez duten elementuak baino ez dira sartzen zerrenda estekatuaren. Zerrendako elementuak errenkada eta zutabearen arabera gorantz ordenatuta daude:



Honako **aziprograma** hau implementatu behar da:

```
public Node {
    Integer balioa;
    Integer errenkada;
    Integer zutabea;
    Node<T> next;
}

public class Matrize {
    Node first;

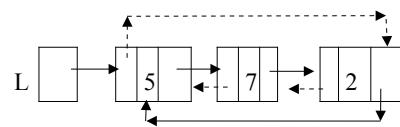
    public Matrize batura(Matrize m1, Matrize m2) {
        // aurre:
        // post: emaitza m1 eta m2 matrizeen batura da
    }
}
```

Erabilitako algoritmoaren **konplexutasun ordenak O(N)** izan behar du, M1 eta M2 matrizeetan 0 ez diren elementuen batezbesteko kopurua N izanda..

Adibidez:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} + \begin{pmatrix} 9 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 9 & 0 & 0 & 0 \\ 0 & 10 & 0 & 5 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

1. Zerrenda biderkatzu (2,5 puntu)



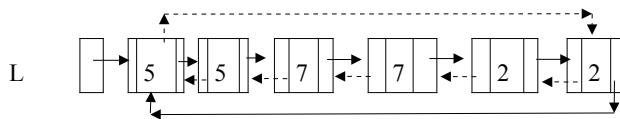
Estekadura bakoitzeko zerrenda zirkularra emanda, ondoko metoda kalkulatu nahi da:

```
public class Node<T> {
    String data;
    Node<T> next;
    Node<T> prev;
}

public class Lista<T> {
    Node<T> first;

    public void biderkatzu(Integer n) {
        // aurre: n >= 1
        // post: zerrenda aldatu da, eta hasierako zerrendako elementu
        //       bakoitza "n" aldiz dauka
    }
}
```

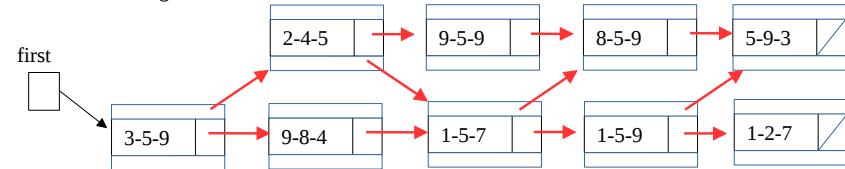
Aurreko adibideko zerrenda emanda, *biderkatzu(2)* deiak ondoko zerrenda bueltatuko luke:



- a) biderkatzu metoda implementatu
- b) Algoritmoaren kostua kalkulatu, **modu arrazoituan**

1. Errobotza (2,5 puntu)

Ondoko lista dugu:



Irudiak errobotzak jarraitu ditzakeen bideak adierazten ditu. Beti une berean hasiko da, eta hortik aurrera errobotak aurrera, ezkerrera edo eskuinera egin dezake. Adibidez, 3-5-9 puntutik hasita hau izango liteke bide bat: (ezkerra, eskuina, aurrera, ezkerra). Zeharkatutako elementuen zerrenda hau izango da: (3-5-9, 2-4-5, 1-5-7, 1-5-9, 5-9-3). Elementu bakoitzak bere koordenatuak adierazten ditu.

lortuKoordenatuak funtzioa implementatu nahi da, errobotak egingo dituen mugimenduak hartuta, zeharkatutako elementuak bueltatzeko:

```
public class Node {
    String koord;
    Node next;
    Node left;
    Node right;
}
public class Bidea {
    Node first;
}

public CircularLinkedList<String> lortuKoordenatuak(
    ArrayList<String> ekintzak)
{
    // pre: "ekintzak" zerrendak gutxienez elementu bat du, eta errobotak
    //       egingo dituen mugimenduak adierazten ditu
    //       Mugimenduak ez dute inoiz errorerik emango (errobotak sekula
    //       ez da saiatuko null erreferentzia bat jarraitzen)
    // post: zerrenda bat bueltatuko du,
    //       errobotak jarraitu dituen koordenaturekin
}

public class CircularLinkedList<T> {
    // zerrenda zirkularra azkenaren erreferentziarekin
    NodeCircularLinkedList<T> last;
}
public class NodeCircularLinkedList<T> {
    T data;
    NodeCircularLinkedList<T> next;
}
```

Adibidez, *lortuKoordenatuak((ezkerra, eskuina, aurrera, ezkerra))* deiak zerrenda hau bueltatuko du: (3-5-9, 2-4-5, 1-5-7, 1-5-9, 5-9-3).

Ondokoa eskatzen da:

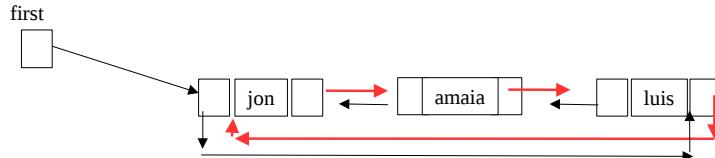
- Algoritmoa implementatu
- Bere kostua eman, modu arrazoituan

Oharra: Bidea eta CircularLinkedList klaseetatik erabiltzen diren metodo guztiak implementatu behar dira.

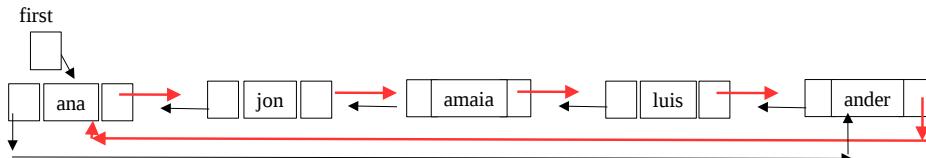
1. Ezabatu azpilista.(1,5 puntu)

Estekadura bikoitzeko bi zerrenda zirkular ditugu:

l1



l2



Metodo hau implementatu nahi dugu:

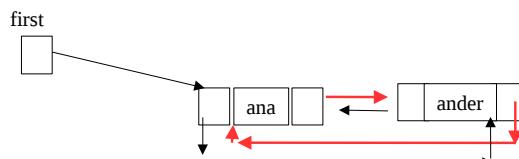
```
public class DoubleNode<T> {
    T data;
    DoubleNode<T> next;
    DoubleNode<T> prev;
}

public class DoubleLinkedList<T> {
    DoubleNode<T> first;

    public void ezabatuLista(DoubleLinkedList<T> azpilista)
    // Aurrebaldintza: "azpilista" uneko listaren zati bat da
    // "azpilista" zerrendako elementuak ordena berean daude lista nagusian
    // Postbaldintza: "azpilista" osoa kendu egin da zerrendatik
}
```

Adibidez, l2.ezabatuLista(l1) deiak zerrenda hau utziko luke:

l2



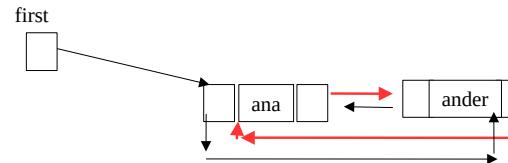
Hau eskatzen da:

- Algoritmoa implementatu
- Kostua kalkulatu modu arrazoituan

1. Gehitu azpilista (1 puntu)

Estekadura bikoitzeko zerrenda zirkular bat dugu:

l



Metodo hau implementatu nahi dugu:

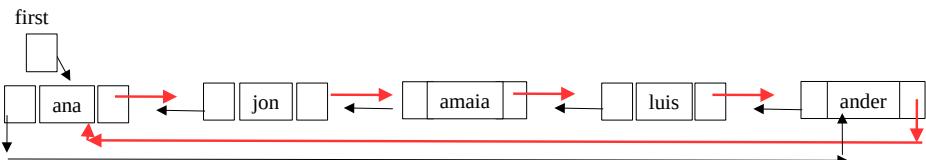
```
public class DoubleNode<T> {
    T data;
    DoubleNode<T> next;
    DoubleNode<T> prev;
}

public class DoubleLinkedList<T> {
    DoubleNode<T> first;

    public void gehitu(T elem, ArrayList<T> berriak)
    // Aurrebaldintza:
    // Postbaldintza: "berriak" zerrendako elementuak listan txertatu dira,
    // "elem" ondoren.
    // "elem" zerrendan ez bailego, lista ez da aldatuko
}
```

Adibidez, l.gehitu("ana", <jon, amaia, luis>) deiak zerrenda hau utziko luke:

l

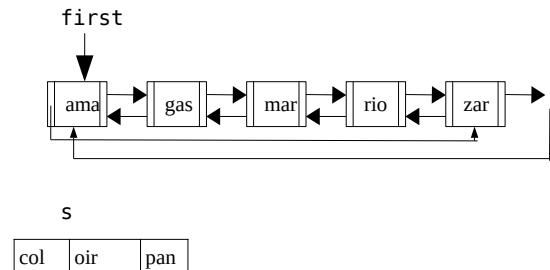


Hau eskatzen da:

- Algoritmoa implementatu
- **Kostua kalkulatu modu arrazoituan**

2. Zerrenda ordenatuak bildu (1 puntu)

Estekadura bikoitzeko zerrenda zirkular bat dugu goranzko ordenan eta array bat (s) ematen digute goranzko ordenan.



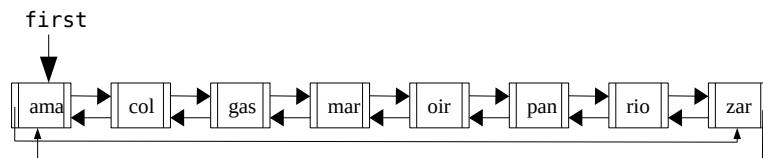
Metodo bat egin nahi da, s array hori emanda, honen elementuak estekadura bikoitzeko zerrenda zirkularrean modu ordenatuan sartuko dituena.

```
public class DoubleNode<T> { // Klase hau ezin da aldatu
    T data;
    DoubleNode<T> prev;
    DoubleNode<T> next;
}

public class DoubleCircularLinkedList { // Estekadura bikoitzeko zerrenda zirkularra ordenan
    DoubleNode<String> first; // ezin da beste atributurik gehitu

    public void sekuentziakFusionatu(String[] s){
        // Aurre: s sekuentzia ordenatuta dago
        // Post: emaitza s-ren balioak estekadura bikoitzeko zerrenda zirkularrean txertatuta izango da (goranzko ordena mantenduko da)
    }
}
```

Goiko 2 sekuentziak emanda sekuentziakFusionatu(s) deiaren emaitza hau izango litzateke:



Oharra: algoritmoaren kostua gehienez bi zerrenden luzerarekiko lineala izan behar da.

2. Blokeen jokoa (2,5 puntu)

Ondoko definizioak emanda:

```
public class Bloke {
    int puntuak;
    int jauzia; // 0 eta 6 arteko balioa
    String norabidea; // 'l' -> ezkerra, 'r' -> eskuina
}

public class Jokoa {
    Stack<Bloke>[] taula; // pilen array-a
    public static int ZUTABEKOP = 7;

    public Jokoa() { // eraikitzailea
        taula = (Stack<Bloke>[]) new Stack[ZUTABEKOP];
        for (int i = 0; i <= ZUTABEKOP - 1; i++) {
            taula[i] = new Stack<Bloke>();
        }
        // blokeen pilak ausaz betetzeko kodea
    }

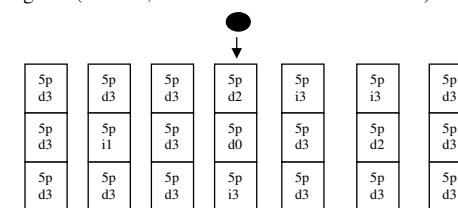
    public int jokatu() {
        // Aurre: jokoa hasieratua izan da (hasierako blokeak sortu dira)
        // Post: partida bat jokatu da. Bola erdiko zutabean hasten da.
        // Emaitza lortutako puntu-kopurua izango da, jokoa gainditu
        // baldin bada, eta -1 bestela
    }
}
```

Klase horiek joko baterako erabiliko dira. Joko horretan bola bat blokeak “apurtzen” joango da. Bola zutabe baten gainean erortzen denean, orduan zutabe horren gainean dagoen blokea “apurtuko” du. Ondoren, bola apurtutako blokeak adierazten duen posiziora joango da, (norabidea, jauzia) bikotearen bidez adierazia. Norabidea ezkerra edo eskuina izan daiteke, eta jauzia 0 eta 6 bitarteko balioa izango da, norabide horretan pasako diren zutabe-kopurua adieraziz. Jokoa bi arrazoi hauengatik bukatu daiteke:

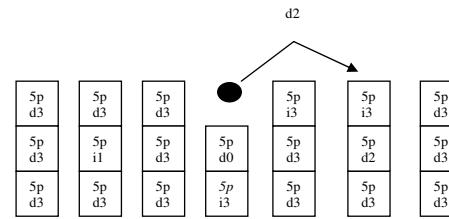
- Bola blokerik gabeko zutabe batean erortzen da. Kasu honetan jokoa gainditu egin da.
- Bola zutabeetatik kanpo erortzen baldin bada, orduan jokoa galdu da.

Ondoko irudiek jokoaren adibide bat erakusten dute:

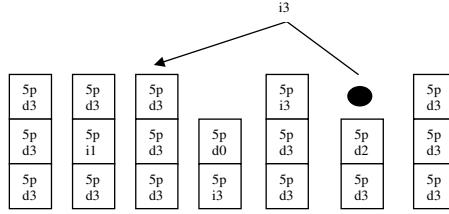
Hasierako egoera (hasteko, erdiko zutabean eroriko da bola):



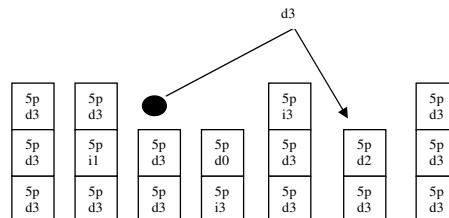
Lehenengo blokean erori eta gero:



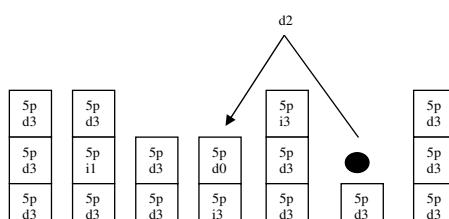
Bolak 2 posizioko jauzia egin du eskuinera.



Blokeak 3 posizioko jauzia eman du ezkerrera.



Blokeak 3 posizioko jauzia eman du eskuinera.



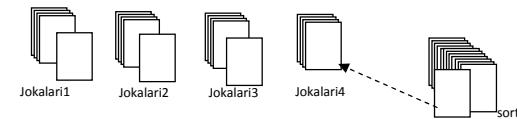
Eta horrela jokoa amaitu arte.

Jokoa amaitzen denean, programak lortutako puntu-kopurua bueltatuko duen Jokatu azpiprograma implementatu behar da.

2. Karta jokaldia (2,5 puntu)

Lau lagunek SEIKOEN jokaldia egin nahi dute. Hauek dira jokoaren ezaugarririk nagusiak:

1. Karta sortak 40 karta ditu, lau sailetan banatuta (urreak, kopak, ezpatak eta bastoia). Sail bakoitzak 10 karta ditu, letek 10era.
2. Kartak lau jokalarien artean banatuko dira, hau da, jokalari bakoitzak 10 karta izango ditu.



3. Partida honela antolatuko da:

- o Lehen jokalariak bere lehen karta hartuko du (gainekoa),
 - Ahal badu (ikus behean), mahaian jarriko du.
 - Ezin badu, orduan karta hori bere karta guztien azpian jarriko du.
- o Eta hurrengo jokalariari pasako zaio txanda.

4. Jokalari batek ondoko kasuetan jar dezake karta mahaian:

- o Karta seiko bat da.
- o Karta ez da seikoa, baina mahaian badagoen karta baten ondorengoa da. Adibidez, nire karta urrezko bikoa baldin bada, eta mahaian urrezko hirukoa balego, edo bestela nire karta urrezko zortzikoa balitz eta mahaian urrezko zapikoa balego.
- Jokoa jokalari batek bere azken karta jartzen duenean amaituko da

Karta_Sorta eta Bilara DMAak erabilita (ez dira implementatu behar DMA hauen eragiketak) ondokoa eskatzen da:

- a) Problema hau ebazteko erabiliko dituzun datu-egiturak definitu (jokalariak eta mahaiaien egorea adierazteko).
- b) **Diseinatu eta idatzi** azpiprograma bat partida bat simulatu eta irabazlea zein den esango diguna.

KartaSorta DMA:

```
public class Karta {
    String palo; // urea, kopa, ezpata, bastoia
    int balioa; // 1 eta 10 arteko balioa
}

public class KartaSorta {
    private Karta[] kartak;

    public KartaSorta () ; // eraikitzailea
    // post: 40 karta daude ausaz

    public Iterator<Carta> iteradore()
    // kartak aztertzeko iteradorea bueltatzen du
}
```

Bilara:

```
public class Bilara<T> {

    public Bilara(); // eraikitzailea
    // bilara hasieratzen du (hutsa)

    public boolean hutsaDa();
    // true B hutsa baldin bada eta false bestela.

    public void txertatuEzk(T elemento);
    // E elementua ezkerretik gehitu dio

    public void txertatuEsk(T elemento);
    // E elementua eskuinetik gehitu dio

    public void ezabatuEzk();
    // ezkerrean dagoen elementua ezabatu du

    public void ezabatuEsk();
    // eskuinean dagoen elementua ezabatu du

    public T lortuEzk();
    // ezkerrean dagoen elementua bueltatzen du

    public T lortuEsk();
    // eskuinean dagoen elementua bueltatzen du
}
```

2. Adierazpen aritmetikoaren ebaluazioa (1,5 puntu)

Djikstrak adierazpen aritmetiko bat ebalutzeko algoritmo bat eman zuen. Adibidez: [\(1+\(\(2+3\)*\(4*5\)\)\)](#)

Metodo hau implementatu behar da:

```
public double ebaluatu(String esp)
    // Aurrebaldintza: Espresio aritmetikoa zuzena da.
    // Adierazpena GUZTIZ parentizatuta dago, hau da,
    // Eragile bakoitzeko, parentesiak daude
    // Eragile guziak bitarrak dira ("X ERAGILE Y" motakoak)
    // Postbaldintza: emaitza adierazpenaren balioa da
```

Aurreko adierazpen hori emanda, emaitza 101 izango da.

Adierazpen aritmetiko parentizatua era honetan ebaluatuko da:

- Bi pila behar dira, bata eragigaien pila, bestea eragileenai
- Sarrerako katearen karaktereak banan-banan aztertuko dira, eta hau egin behar da pauso bakoitzean:
 - Ezkerreko parentesia bada, ezer ez
 - Eragaiaga bada, orduan eragigaien pilan sartu
 - Eragilea baldin bada, orduan eragileen pilan sartu
 - Eskuineko parentesia baldin bada, orduan:
 - hartu bi eragigai eragigaien pilatik
 - hartu eragilea eragileen pilatik
 - kalkulatu emaitza eta eragileen pilan sartu

Adibidea

Kate hau emanda: [\(1+\(\(2+3\)*\(4*5\)\)\)](#)

Eragileen pilai

Eragigaien pilai

- Lehen parentesia “(“ → ezer ez

Katea: [1+\(\(2+3\)*\(4*5\)\)\)](#)

- “1” → sartu eragigaien pilan

Eragileen pilai

1

Katea: [+\(\(2+3\)*\(4*5\)\)\)](#)

- “+” → sartu eragileen pilan

Eragileen pilai

+

Eragigaien pilai

Katea: $((2+3)*(4*5))$

- parentesia “(“ → ezer ez
- parentesia “(“ → ezer ez

Katea: $2+3)*(4*5))$

- “2” → sartu eragigaien pilan

Eragileen pila	<table border="1"><tr><td>+</td></tr></table>	+
+		
Eragigaien pila	<table border="1"><tr><td>1 2</td></tr></table>	1 2
1 2		

Katea: $+3)*(4*5))$

- “+” → sartu eragileen pilan

Eragileen pila	<table border="1"><tr><td>++</td></tr></table>	++
++		
Eragigaien pila	<table border="1"><tr><td>1 2</td></tr></table>	1 2
1 2		

Katea: $3)*(4*5))$

- “3” → sartu eragigaien pilan

Eragileen pila	<table border="1"><tr><td>++</td></tr></table>	++
++		
Eragigaien pila	<table border="1"><tr><td>1 2 3</td></tr></table>	1 2 3
1 2 3		

Katea: $)*(4*5))$

- “)” → eragiketa egin

Eragileen pila	<table border="1"><tr><td>+</td></tr></table>	+
+		
Eragigaien pila	<table border="1"><tr><td>1 5</td></tr></table>	1 5
1 5		

Katea: $*(4*5))$

- “*” → sartu eragileen pilan

Eragileen pila	<table border="1"><tr><td>+ *</td></tr></table>	+ *
+ *		
Eragigaien pila	<table border="1"><tr><td>1 5</td></tr></table>	1 5
1 5		

Katea: $(4*5))$

- parentesia “(“ → ezer ez

Katea: $4*5))$

- “4” → sartu eragigaien pilan

Eragileen pila	<table border="1"><tr><td>+ *</td></tr></table>	+ *
+ *		
Eragigaien pila	<table border="1"><tr><td>1 5 4</td></tr></table>	1 5 4
1 5 4		

Katea: $*5))$

- “*” → sartu eragileen pilan

Eragileen pila	<table border="1"><tr><td>+ * *</td></tr></table>	+ * *
+ * *		
Eragigaien pila	<table border="1"><tr><td>1 5 4</td></tr></table>	1 5 4
1 5 4		

Katea: $5))$

- “5” → sartu eragigaien pilan

Eragileen pila	<table border="1"><tr><td>+ * *</td></tr></table>	+ * *
+ * *		
Eragigaien pila	<table border="1"><tr><td>1 5 4 5</td></tr></table>	1 5 4 5
1 5 4 5		

Katea: $))$

- “)” → eragiketa egin

Eragileen pila	<table border="1"><tr><td>+ *</td></tr></table>	+ *
+ *		
Eragigaien pila	<table border="1"><tr><td>1 5 20</td></tr></table>	1 5 20
1 5 20		

Katea: $)$

- “)” → eragiketa egin

Eragileen pila	<table border="1"><tr><td>+</td></tr></table>	+
+		
Eragigaien pila	<table border="1"><tr><td>1 100</td></tr></table>	1 100
1 100		

Katea: $)$

- “)” → eragiketa egin

Eragileen pila	<table border="1"><tr><td></td></tr></table>	
Eragigaien pila	<table border="1"><tr><td>101</td></tr></table>	101
101		

Emaitza eragigaien pilan dago!

Algoritmoa implementatu, eta bere kostua kalkulatu, modu arrazoituan.

Oharra: zenbaki baten balioa kalkulatzeko, ondoko metodoa erabil daiteke (ez da implementatu behar):

```
public Double parse(String s)
// Post: zenbaki bat daukan string bat emanda, balioa bueltatzen du.
```

2. Koloreen jokoa (1,5 puntu)

Joko bateko 5 jokalarik kolore bateko fitxa-kopuru bat dute (jokalari guztiekin hasten dute jokoa kopuru berarekin). Jokalari bakoitzak hasieran kolore bereko fitxak izango ditu, eta jokalari guztiekin kolore desberdinak dituzte. Gainera, bankua kolore beltzeko fitxak ditu (bankua zero garren jokalaria izango da). Jokoak mahai bat behar du, bertan fitxak ondoren azaltzen diren arauen arabera jarriko direlarik. Irudi honek jokoaren hasierako egoera erakusten du:

j0 (bankua)	
j1	
j2	
j3	
j4	
j5	

Jokaldi bakoitzan bi dado botako dira, ondoko arauetan:

1. Lehen dadoaren balioa 6 baldin bada, jokoa amaitzen da.
2. Bestela, lehen dadoa bikoitia baldin bada, bigarren dadoaren balioak esango du zein jokalarik mugitzen duen fitxa bat mahaira. Dadoaren balioa 1etik 6ra inokoa denez, balio horri bat kenduko zaio [0, 5] tarteko balioa lortzeko. Jokalariak fitxarik ez baleuka, ez da ezer egingo.
3. Lehen dadoa bakoitia baldin bada, fitxa bat mugituko da mahaitik bigarren dadoaren balioak esango duen jokalarira. Fitxa hori jokalariak erabiliko duen azkena izango da. Mahaian fitxarik ez balego, ez da mugimendurik egingo.
4. Irabazlea fitxa beltz gehien duen jokalaria izango da (bankua kontuan hartu gabe). Berdinketaren kasuan, irabazlea puntuazio maximoa duen lehen jokalaria izango da.

Adibidez, hasierako egoeran dadoek (4, 3) emango balute, bigarren jokalariaren fitxa bat hartuko litzateke (3 ken 1) eta mahaian jarri beharko litzateke:

j0 (bankua)	
j1	
j2	
j3	
j4	
j5	

mahaia

mahaia

Dadoak berriro botatzen direnean (4, 1) bikotea aterako balitz, fitxa bat mugituko da 0garren jokalaritik mahaira:

j0 (bankua)	
j1	
j2	
j3	
j4	
j5	

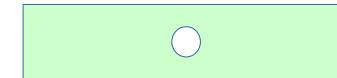
mahaia



Eta orain (3, 5) aterako balitz, lehenengo dado bakoitia denez, fitxa bat mugituko da mahaitik 4garren jokalarira. Kontuan izan mugituko den fitxa mahaian kokatu zen azkenetakoa izango dela. Gainera, fitxa hori 4garren jokalariak jokatuko duen azkena izango da.

j0 (bankua)	
j1	
j2	
j3	
j4	
j5	

mahaia



Azpiprograma hau implementatu behar da:

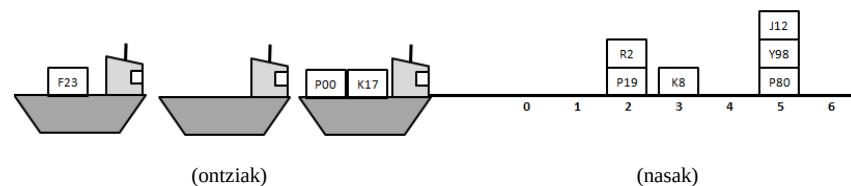
```
public class Jokoa {
    Queue<Integer>[] jokalariek;
    // Fitxen koloreak balio osokoan bidez adierazten dira: beltzak 0 eta
    // beste jokalarien kolorea bere posizioarekin bat eterriko da (hau da,
    // 1 jokalariak 1 kolorea, ...)

    Stack<Integer> mahaia;

    public int jokoa(int n, ArrayList<Jokaldi> jokaldiak) {
        // aurre: n jokalari bakoitzaren hasierako fitxa-kopurua da
        // "jokaldiak" zerrendak partida bateko jokaldiak ditu
        // post: emaitza irabazlearen zenbakia da
    }
    public class Jokaldi {
        int dado1;
        int dado2;
    }
}
```

2. ariketa: Portua (1,5 puntu)

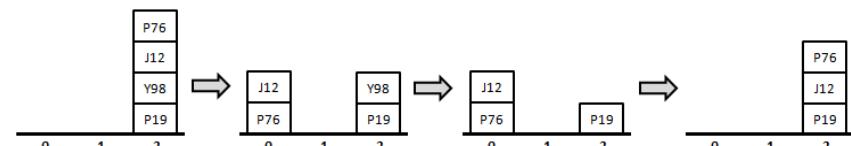
Portu bateko jardueraren simulazioa egin nahi dugu. Portuak hainbat nasa ditu kontainerrak pilatzeko. Eguna hasten denean, ontziak ilaran sartzen dira portura, nasetatik kontainerrak kargatzeko edo nasetan kontainerrak deskargatzeko.



Ontzi bakoitzak kontainerrak kargatzeko eskaerak izango dituzte, eta kargago ontziek nasetatik kontainerrak kargatzekoak. Badakigu ere 0 zenbakia duen nasa berezia dela eta ez dela eskaeretan agertuko.

Horrela funtzionatuko du jardueraren simulazioak, ontzien ilaran ontziak dauden bitartean:

- ◆ Ilarako lehen ontzia hartuko da.
 - ◆ Deskargako ontzi bat bada:
 - Ontziaren lehenengo *maxEskera* eskaerak kudeatuko dira (gutxiago baditu, dituen guztiak):
 - Eskaera bakoitzeko: kontainera adierazitako nasan utzik da.
 - *maxEskera* eskaera baino gehiago bazituen, ontzia ontzien ilaran jarriko da berriz, ilararen amaieran.
 - ◆ Kargako ontzi bat bada:
 - Ontziaren eskaera guztiak kudeatuko dira:
 - Eskaera bakoitzeko: kontainera adierazitako nasatik kendu behar da. Kontainer bat kendu behar denean, 0. nasa erabili dezakegu aldi baterako bere gainean dauden kontainerrak uzteko, eta behin gure kontainerra kendu dugula, 0. nasan utzi ditugunak jatorrizko nasan utzik ditugu berriz. Adibidez, ondorengo irudiko Y98 kontainerra kentzeko:



Portua klasearen portuaSimulatu metodoa implementatzea eskatzen da.

```
public class Eskaera {
    String kontainerKode;
    int nasa;
}

public class Ontzia {
    int mota; //0 (deskarga), 1 (karga)
    String izena;
    Queue<Eskaera> eskaerak;
}

public class Portua {

    Stack<String> nasak[];

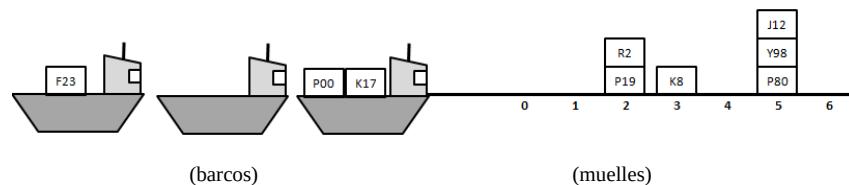
    public void portuaSimulatu(Queue<Ontzia> ontziak, int maxEskaera, int nasaKop) {
        //Aurre: maxEskaera deskargako ontzi batek txanda batean kudeatu ditzakeen
        //eskaera kopurua da (>=1)
        //Aurre: nasaKop portuaaren nasa kopurua da (>=2). 0. nasa berezia da.
        //Post: portuaaren jarduera simulatu da, ontzien eskaerak betez.

        //Nasak sortu
        ...

        //Portuaaren jarduera simulatu
        ...
    }
}
```

Ejercicio 2: Puerto (1,5 puntos)

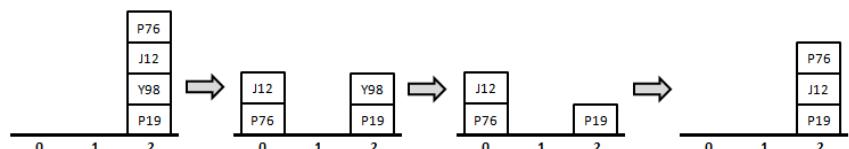
Queremos simular la actividad de un puerto. El puerto tiene diferentes muelles para guardar los contenedores que llegan. Cuando empieza el día, los barcos entran ordenadamente al puerto para cargar y descargar contenedores a y desde los muelles.



Cada barco es o bien de carga o de descarga, y tiene una serie de peticiones a completar. Cada petición tiene un código de contenedor y un número de muelle. Los barcos de descarga tendrán peticiones para descargar contenedores en el muelle, y los de carga para cargar contenedores desde el muelle. El muelle de código 0 es especial y no aparecerá en las peticiones.

La simulación de la actividad del puerto funciona de la siguiente manera, hasta que se atiendan todas las peticiones de los barcos:

- ◆ Se atenderá el primer barco que ha llegado.
- ◆ Si es un barco de descarga:
 - Se atenderán las primeras *maxPeticiones* de un barco (si tiene menos, todas)
 - Para cada petición: se dejará el contenedor en el muelle especificado.
 - Si el barco tuviera más de *maxPeticiones*, se colocará el barco al final de la cola con las peticiones restantes.
- ◆ Si es un barco de carga:
 - Se atenderán todas las peticiones del barco:
 - Para cada petición: el contenedor se cogerá del muelle especificado. Para sacar un contenedor, se puede usar el muelle 0 para dejar ahí temporalmente los contenedores que se encuentran encima del contenedor pedido y, una vez quitado ese contenedor, se volverán a colocar en ese muelle los contenedores apilados temporalmente en el muelle 0. Por ejemplo, en la siguiente figura se muestra cómo se cargaría en el barco el contenedor Y98:



Se pide implementar el método *simularPuerto* de la clase *Puerto*.

```

public class Peticion {
    String codigoDeContenedor;
    int muelle;
}

public class Barco {
    int tipo; // 0 (descarga), 1 (carga)
    String nombre;
    Queue<Peticion> peticiones;
}

public class Puerto {
    Stack<String>[] muelles;

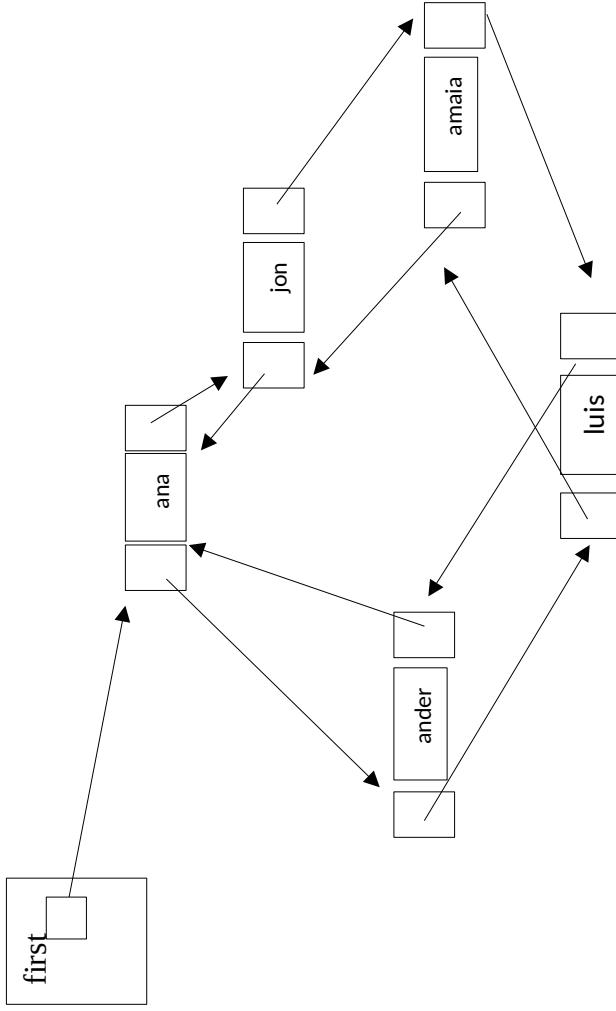
    public void simularPuerto(Queue<Barco> barcos, int maxPeticiones, int numMuelles){
        // Pre: maxPeticiones es el número máximo de peticiones que se pueden atender en
        //       el turno de un barco
        // Pre: numMuelles es el número de muelles del puerto (>=2). El muelle 0 es especial
        // Post: se ha simulado la actividad del puerto, atendiendo las peticiones de los barcos

        // crear los muelles
        ...

        // simular la actividad del puerto
        ...
    }
}

```

1. (1,5 puntu) Estekadura bikoitzeko zerrenda zirkularra dugu:



Funtzio hau implementatu nahi dugu:

```

public T azkenaLortu(Integer jauzia)
{
    public DoubleNode<T> {
        T data;
        DoubleNode<T> next;
        DoubleNode<T> prev;
    }

    public DoubleLinkedList<T> {
        DoubleNode<T> first;
        public T azkenaLortu(Integer jauzia)
        {
            // aurre: listak gutxienetako elementu bat du
            // post: elementu bakarreko lista bueltatuko du, zerrendatik pausu
            // bakoitzean elementu bat kenduz, unekotik "jauzi" elementu pasatuz
            // (erlojuko orratzen norabidean) elementu bakarra geratu arte
        }
    }
}
  
```

Adibidez, azkenaLortu(4) deiaak "jon" bueltatuko du (elementu hauek ezabatu ondoren:
ander, luis, ana eta amaya).

Hau eskatzen da:

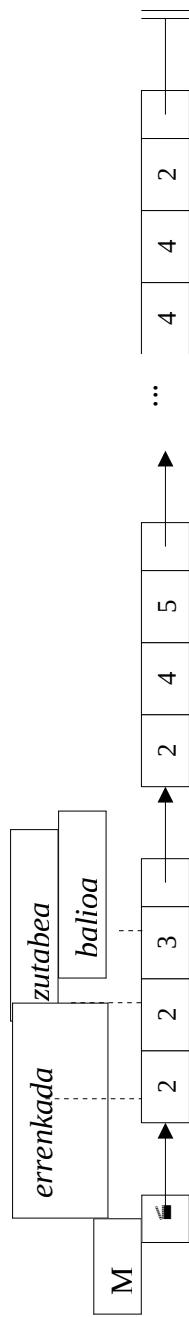
- Algoritmoa implementatu
- Kostua kalkulatu, modu arrazoituan.

1.- Matrize batuketa (2,5 puntu)

Matrize sakabanatueta posizio gehienetan zero balioa egoten da. Adibidez:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

Mota honetako matrizea zerrenda estekatu sinplea erabiliz adierazi daitake. Matrizean 0 balioa ez duten elementuak baino ez dira sartzen zerrenda estekatuau. Zerrendako elementuak errenkada eta zutabearen arabera gorantz ordenatuta daude:



Honako azpiprograma hau implementatu behar da:

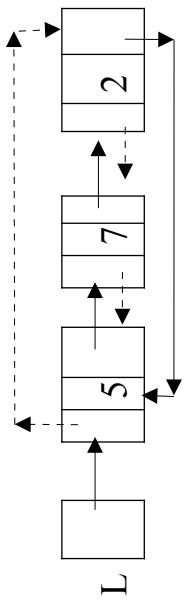
```
public Node {  
    Integer balioa;  
    Integer errenkada;  
    Integer zutabea;  
    Node<T> next;  
}  
  
public class Matrize {  
  
    Node first;  
  
    public Matrize batura(Matrize m1, Matrize m2) {  
        // aurre:  
        // post: emaitza m1 eta m2 matrizeen batura da  
    }  
}
```

Erabilitako algoritmoaren **komplexutasun ordenak O(N)** izan behar du, M1 eta M2 matrizeetan 0 ez diren elementuen batezbesteko kopurua N izanda..

Adibidez:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} + \begin{pmatrix} 9 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 9 & 0 & 0 & 0 \\ 0 & 10 & 0 & 5 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

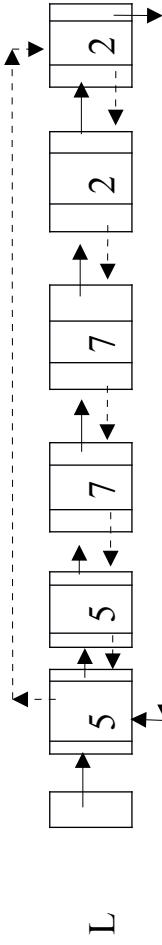
1. Zerrenda biderkatu (2,5 puntu)



Estekadura bikoitzeko zerrenda zirkularra emanda, ondoko metodoa kalkulatu nahi da:

```
public class Lista<T> {
    public Node<T> first;
    public void biderkatu(Integer n) {
        // aurre: n >= 1
        // post: zerrenda aldatu da, eta hasierako zerrendako elementu
        // bakoitzak "n" aldiz dauka
    }
}
```

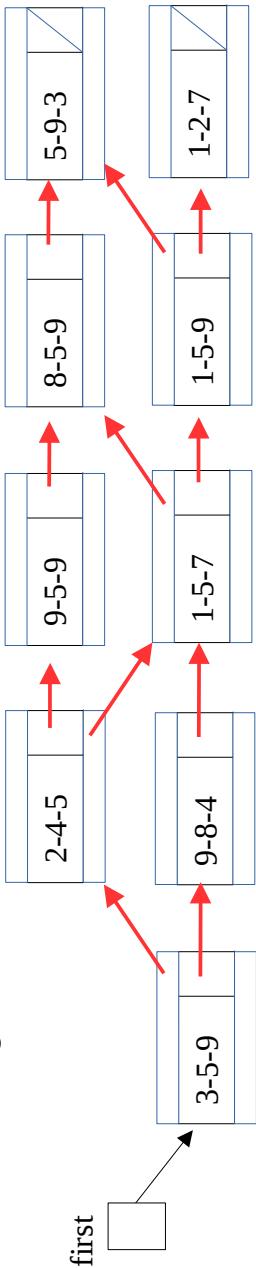
Aurreko adibideko zerrenda emanda, *biderkatu(2)* deik ondoko zerrenda bueltatuko luke:



- a) biderkatu metodoa implementatu
- b) Algoritmoaren kostua kalkulatu, **modu arrazoituan**

1. Erroba (2,5 puntu)

Ondoko lista dugu:



Iridiak errobot batek jarraitu dituzkaen bideak adierazten ditu. Befi une berean hasiko da, eta hortik aurrera errobotak aurera, ezkerrean edo eskuinera egin dezake. Adibidez, 3-5-9 puntutik hasita hau izian liteke bide bat: (ezkerra, eskuina, aurera, ezkerra). Zeharkatutako elementuen zerrenda hau izango da: (3-5-9, 2-4-5, 1-5-7, 1-5-9, 5-9-3). elementu bakoitzak bere koordenatuak adierazten ditu.

Iortu Koordenatuak funtzia implementatu nahi da, errobotak egingo dituen mugimenduak hartuta, zeharkatutako elementuak bueltatzeko:

```
public class Node {  
    String koord;  
    Node next;  
    Node left;  
    Node right;  
}  
  
public class Bidea {  
    Node first;  
}  
  
public CircularLinkedList<String> lortukoordenatuak(  
    ArrayList<String> ekintzak)  
    // pre: "ekintzak" zerrendak gutxienez elementu bat du, eta errobotak  
    // egingo dituen mugimendua adierazten ditu  
    // Mugimenduek ez dute inoiz errorerik emango (errobotak sekula  
    // ez da saiatuko null erreferentzia bat jarraitzen)  
    // post: zerrenda bat bueltatuko du,  
    // errobotak jarraitu dituen koordenatuakin  
}
```

```
public class CircularLinkedList<T> {  
    // zerrenda zirkularra azkenaren erreferentziarekin
```

```
} public class NodeCircularLinkedList<T> {  
    T data;  
    NodeCircularLinkedList<T> next;
```

Adibidez, lortu Koordenatuak (ezkerra, eskuina, aurra, ezkerra) deiakeren zerranetan bueltatuko
du: (3-5-9, 2-4-5, 1-5-7, 1-5-9, 5-9-3).

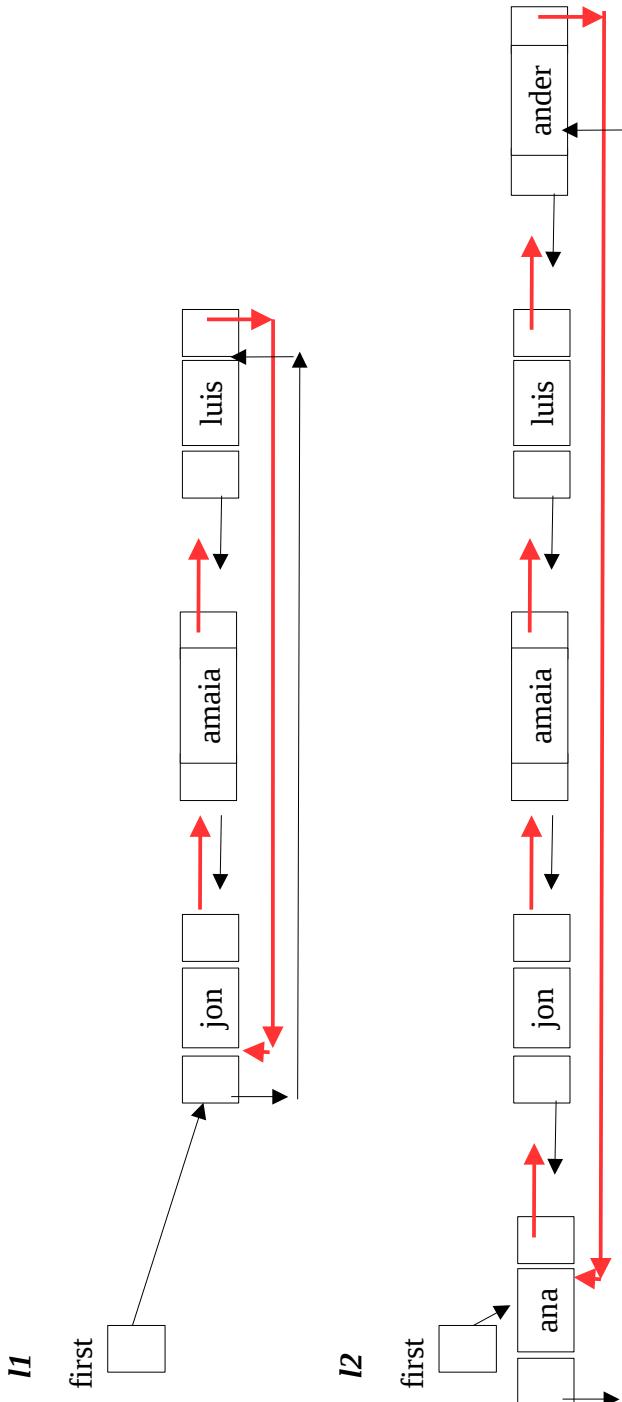
Ondokoak eskatzen da:

- Algoritmoa implementatu
 - Bere kostua eman, modu arrazoituaren

Oharra: Bidea eta CircularLinkedList klaseetako erabilten diren metodo guztiak implementatu behar dira.

1. Ezabatu azpilista.(1,5 puntu)

Estekadura bikoitzeko bi zerrenda zirkular ditugu:



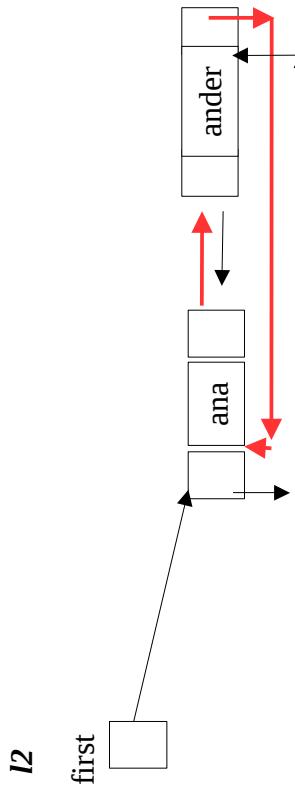
Metodo hau implementatu nahi dugu:

```
public class DoubleNode<T> {
    T data;
    DoubleNode<T> next;
    DoubleNode<T> prev;
}

public class DoubleLinkedList<T> {
    DoubleNode<T> first;
}

public void ezabatuLista(DoubleLinkedList<T> azpilista)
{
    // Aurrebalrintza: "azpilista" uneko listaren zati bat da
    // "azpilista" zerrendako elementuak ordena berean daude lista nagusian
    // Postbalrintza: "azpilista" osoa kendu egin da zerrendatik
}
```

Adibidez, *l2.ezabatuLista(l1)* deiak zerrenda hau utzikо luke:



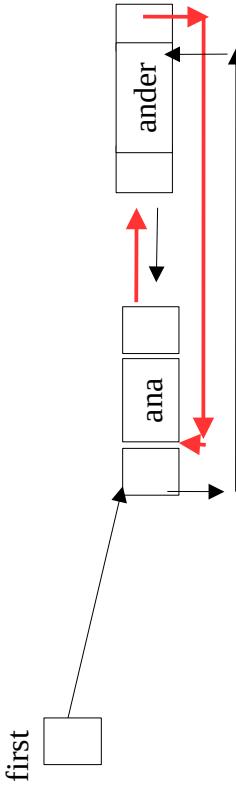
Hau eskatzen da:

- Algoritmoa implementatu
- Kostua kalkulatu modu arrazoituan

1. Gehitu azpilista (1 puntu)

Estekadura bikoitzeko zerrenda zirkular bat dugu:

I



Metodo hau implementatu nahi dugu:

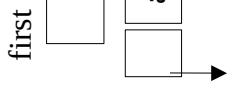
```
public class DoubleNode<T> {
    T data;
    DoubleNode<T> next;
    DoubleNode<T> prev;
}

public class DoubleLinkedList<T> {
    DoubleNode<T> first;

    public void gehitu(T elem, ArrayList<T> berriak)
        // Aurrebalintza:
        // Postbalintza:
        // "berriak" zerrendako elementuak listan txertatu dira,
        // "elem" ondoren.
        // "elem" zerrendan ez bailego, lista ez da aldatuko
}
}
```

Adibidez, I.gehitu("ana", <jon, amai, luis>) deiakerrenda hau utzikor luke:

I

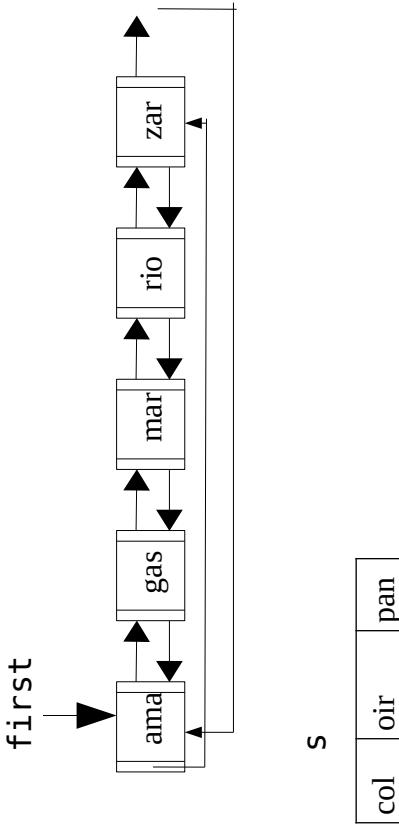


Hau eskatzen da:

- Algoritmoa implementatu
- Kostua kalkulatu modu arrazoituan

2. Zerrenda ordenatuak bildu (1 puntu)

Estekadura bikoitzeko zerrenda zirkular bat dugu goranzko ordenan eta array bat (s) ematen digute goranzko ordenan.



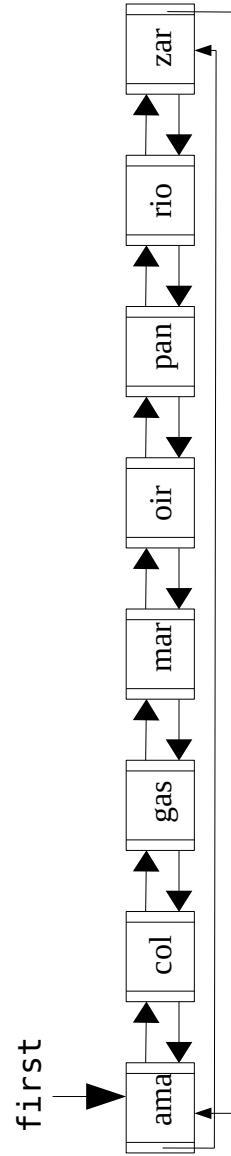
Metodo bat egin nahi da, s array hori emanda, honen elementuak estekadura bikoitzeko zerrenda zirkularrean modu ordenatuan sartuko dituena.

```
public class DoubleNode<T> { // Klase hau ezin da aldatu
    T data;
    DoubleNode<T> prev;
    DoubleNode<T> next;
}

public class DoubleCircularLinkedList { // Estekadura bikoitzeko zerrenda zirkularra ordenan
    DoubleNode<String> first; // ezin da beste atributurik gehitu

    public void sekuentziakFusionatu (String[] s){
        // Aurre: s sekuentzia ordenatuta dago
        // Post: emaitza s-ren balioak estekadura bikoitzeko zerrenda zirkularrean txertatuta
        // izango da (goranzko ordena mantenduko da)
    }
}
```

Goiko 2 sekuentziak emanda sekuentziakFusionatu(s) deiaaren emaitza hau izango lizateke:



Oharra: algoritmoaren kostua gehienez bi zerrenden luzerarekiko lineala izan behar da.

2. Blokeen jokoak (2,5 puntu)

Ondoko definizioak emanda:

```
public class Bloke {
    int puntuak;
    int jauzia; // 0 eta 6 arteko balioa
    String norabidea; // '1' -> ezkerria, 'r' -> eskuina
}

public class Jokoak {
    Stack<Bloke>[] taula; // pilen array-a
    public static int ZUTABEKOP = 7;

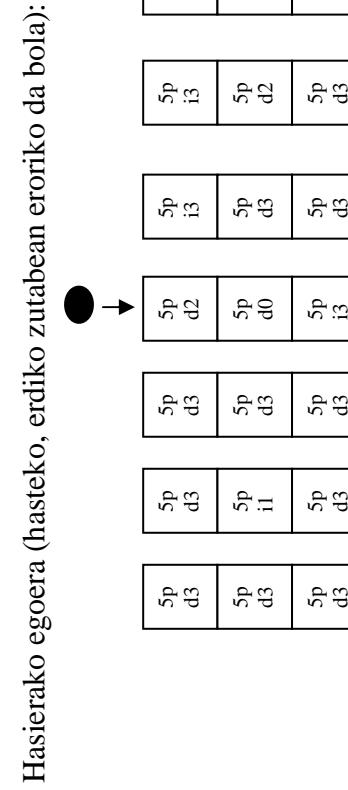
    public Jokoak() { // eraikitzailea
        taula = (Stack<Bloke>[]) new Stack[ZUTABEKOP];
        for (int i = 0; i <= ZUTABEKOP - 1; i++) {
            taula[i] = new Stack<Bloke>();
        }
        // blokeen pilak ausaz betetzeko kodea
    }

    public int jokatu() {
        // Aurre: jokoak hasieratua izan da (hasierako blokeak sortu dira)
        // Post: partida bat jokatu da. Bola erdiko zutabean hasten da.
        // Emaitzta lortutako puntu-kopurua izango da, jokoak gainditu
        // baldin bada, eta -1 bestela
    }
}
```

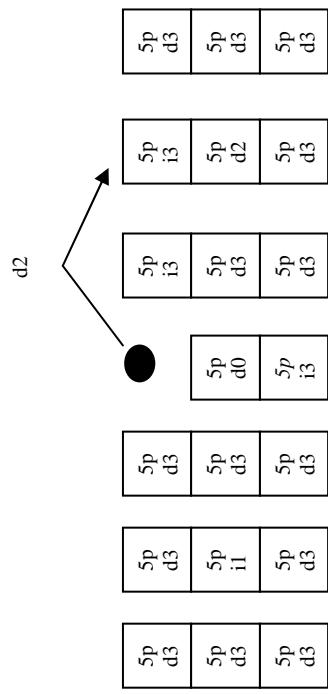
Klase horiek joko baterako erabiliko dira. Joko horretan bola bat blokeak “apurtzen” joango da. Bola zutabe baten gainean erortzen denean, orduan zutabe horren gainean dagoen blokea “apurtuko” du. Ondoren, bola apurtutako blokeak adierazten duen posiziorea joango da, (norabidea, jauzia) bikotearen bidez adierazia. Norabidea ezkerria edo eskuina izan daiteke, eta jauzia 0 eta 6 bitarteko balioa izango da, norabide horretan pasako diren zutabe-kopurua adieraziz. Jokoak bi arrazoi hauengatik bukatu daiteke:

- Bola blokerik gabeko zutabe batean erortzen da. Kasu honetan jokoak gainditu egin da.
- Bola zutabeetatik kanpo erortzen baldin bada, orduan jokoak galdu da.

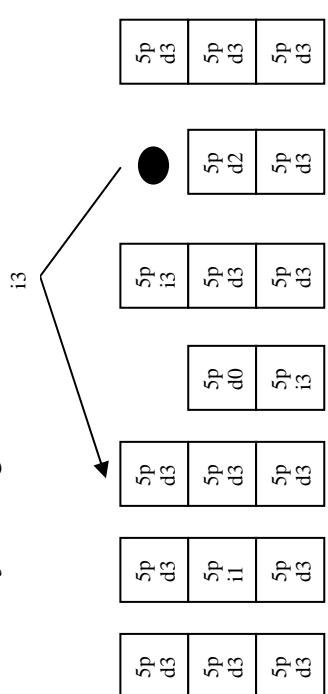
Ondoko irudiek jokoaren adibide bat erakusten dute:



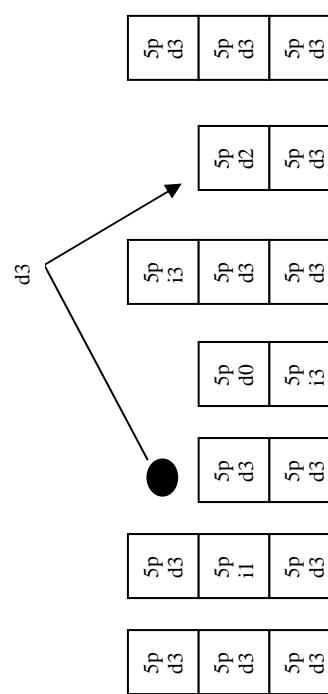
Lehenengo blokean erori eta gero:



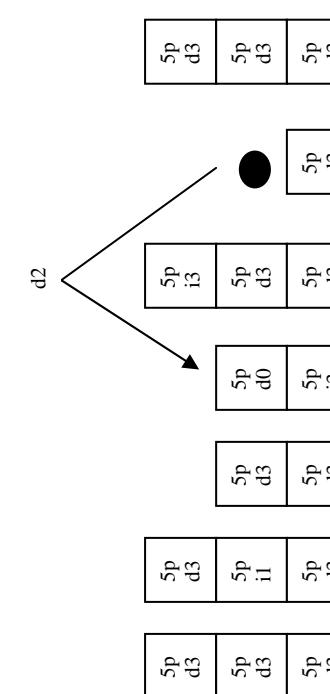
Bolak 2 posizioko jauzia egin du eskuinera.



Bloakeak 3 posizioko jauzia eman du ezkerrera.



Bloakeak 3 posizioko jauzia eman du eskuinera.



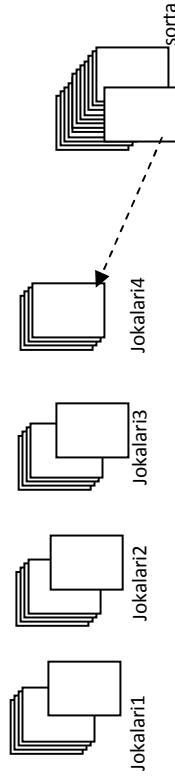
Eta horrela jokoaa amaitu arte.

Jokoaa amaitzen denean, programak lortutako puntu-kopurua bueltatuko duen Jokatu aziprograma implementatu behar da.

2. Karta jokaldia (2,5 puntu)

Lau lagunek SEIKOEN jokaldia egin nahi du. Hauek dira jokoaren ezagarrin nagusiak:

1. Karta sortak 40 karta ditu, lau sailetan banatuta (utreak, kopak, ezpatak eta bastoiak). Sail bakoitzak 10 karta ditu, 1 etik 10era.
2. Kartak lau jokalarien artean banatuko dira, hau da, jokalari bakoitzak 10 karta izango ditu.



3. Partida honela antolatuko da:
 - o Lehen jokalariak bere lehen karta hartuko du (gainekoa),
 - Ahal badu (ikus behean), mahaian jarriko du.
 - Ezin badu, orduan karta hori bere karta guztieng azpian jarriko du
 - o Eta hurrengo jokalariari pasako zaio txanda.
 4. Jokalari batek ondoko kasuetan jar dezake karta mahaian:
 - o Karta seiko bat da.
 - o Karta ez da seikoa, baina mahaian badagoen karta baten ondorengoa da. Adibidez, nire karta urezko bikoa baldin bada, eta mahaian urezko hirukoa balego, edo bestela nire karta urezko zortzikoan balitz eta mahaian urezko zazpikoa balego.
- Jokoa jokalari batek bere azken karta jartzen duenean amaituko da
- Karta_Sorta eta Bilara DMAak erabilita (ez dira implementatu behar DMA hauen eragiketak) ondokoak eskatzen da:
- a) Problema hau ebazteko erabiliko dituzun datu-egiturak definitu (jokalariak eta mahaian egoera adierazteko).
 - b) *Diseinatu eta idatzi* azpiprograma bat partida bat simulatu eta irabazlea zein den esango diguna.

KartaSorta DMA:

```

public class Karta {
    String palo; // urea, kopa, ezpata, bastoia
    int balioa; // 1 eta 10 arteko balioa
}

public class Kartasorta {
    private Karta[] kartak;

    public Kartasorta () ; // eraikitzailea
    // post: 40 karta daude ausaz

    public Iterator<Carta> iteradore()
    // kartak aztertzeko iteradorea bueltatzen du
}

```

Bilara:

```

public class Bilara<T> {

    public Bilara(); // eraikitzailea
    // bilara hasieratzen du (hutsa)

    public boolean hutsa();
    // true B hutsa baldin bada eta false bestela.

    public void txertatuEzk(T elemento);
    // E elementua ezkerretik gehitu dio

    public void ezabatuEzk();
    // ezkerrean dagoen elementua ezabatu du

    public void ezabatuEzk();
    // eskuinean dagoen elementua ezabatu du

    public T lortuezk();
    // ezkerrean dagoen elementua bueltatzen du

    public T lortuezk();
    // eskuinean dagoen elementua bueltatzen du
}

```

2. Adierazpen aritmetikoaren evaluazioa (1,5 puntu)

Dikstrak adierazpen aritmetiko bat evaluatzeko algoritmo bat eman zuen. Adibidez: (1+((2+3)*(4*5)))

Metodo hau implementatu behar da:

```
public double evaluatu(String esp)
{
    // Aurrealdintza: Espresio aritmetikoa zuzena da.
    // Adierazpena GUZTIZ parentizatuta dago, hau da,
    // Eragile bakoitzeko, parentesiak daude
    // Eragile guztia bitarrak dira ("X ERAGILE Y" motakoak)
}

// Postbaldintza: emaitza adierazpenaren balioa da
```

Aurreko adierazpen hori emanda, emaitza 101 izango da.

Adierazpen aritmetiko parentizatua era honetan evaluatuko da:

- Bi pila behar dira, bata eragaien pila, bestea eragileenena
- Sarrerako katearen karaktereak banan-banan aztertuko dira, eta hau egin behar da pauso bakoitzean:
 - Ezkerreko parentesia bada, ezer ez
 - Eragigai a bida, orduan eragigaien pilan sartu
 - Eragilea baldin bida, orduan eragileen pilan sartu
 - Eskuiñeko parentesia baldin bida, orduan:
 - hartu bi eragigaien pilatik
 - hartu eragilea eragileen pilatik
 - kalkulu emaitza eta eragileen pilan sartu

Adibidea

Kate hau emanda: (1+((2+3)*(4*5)))

Eragileen pila
Eragigaien pila

- Lehen parentesia “(“ → ezer ez

Katea: 1+((2+3)*(4*5))

- “1” → sartu eragigaien pilan

Eragileen pila
Eragigaien pila

Katea: +((2+3)*(4*5))

- “+” → sartu eragileen pilan

+ 1
Eragigaien pila

Katea: ((2+3)*(4*5))

- parentesia “(“ → ezer ez
- parentesia “(“ → ezer ez

Katea: 2+3)*(4*5))

- “2” → sartu eragaien pilan

Eragileen pila	+
Eragaien pila	1 2

Katea: +3)*(4*5))

- “+” → sartu eragileen pilan

Eragileen pila	++
Eragaien pila	1 2

Katea: 3)*(4*5))

- “3” → sartu eragaien pilan

Eragileen pila	+
Eragaien pila	1 2 3

Katea:)*(4*5))

- “)” → eragiketa egin

Eragileen pila	+
Eragaien pila	1 5

Katea: *(4*5))

- “*” → sartu eragileen pilan

Eragileen pila	+ *
Eragaien pila	1 5

Katea: (4*5))

- parentesia “(“ → ezer ez

Katea: 4*5))

- “4” → sartu eragaien pilan

Eragileen pila	+
Eragaien pila	1 5 4

Katea: *5))

- “*” → sartu eragileen pilan

Eragileen pila	+ **
Eragaien pila	1 5 4

Katea: 5))

- “5” → sartu eragigaien pilan

Eragileen pila	+ * *	
Eragigaien pila	1 5 4 5	

Katea:)))

- “)” → eragiketa egin

Eragileen pila	+ *	
Eragigaien pila	1 5 20	

Katea:))

- “)” → eragiketa egin

Eragileen pila	+ *	
Eragigaien pila	1 100	

Katea:)

- “)” → eragiketa egin

Eragileen pila	+ *	
Eragigaien pila	101	

Emaitzia eragigaien pilan dago!

Algoritmoa implementatu, eta bere kostua kalkulatu, modu arrazoiuan.

Oharra: zenbaki baten balioa kalkulatzeko, ondoko metodoa erabil daiteke (ez da implemenatu behar):

```
public Double parse(String s)
// Post: zenbaki bat daukan string bat emanda, balioa bueltatzen du.
```

2. Koloreen jokoa (1,5 puntu)

Joko bateko 5 jokalarik kolore bateko fitxa-kopuru bat dute (jokalarri guztiak hasten dute jokoa kopuru berarekin). Jokalarri bakoitzak hasieran kolore bereko fitxak izango ditu, eta jokalarri guztiekin kolore desberdinak dituzte. Gainera, bankua kolore beltzezko fitxak ditu (bankua zeroaren jokalarria izango da). Jokoak mahai bat behar du, bertan fitxak ondoren azaltzen diren arauen arabera jarriko direlarik. Irudi honek jokoaren hasierako egoera erakusten du:

j0 (bankua)					
j1					
j2					
j3					
j4					
j5					

Jokaldi bakoitzan bi dado botako dira, ondoko arauetan:

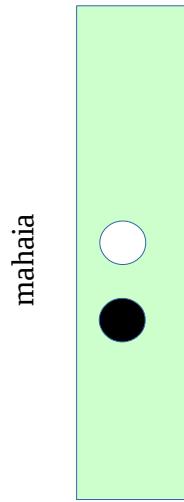
1. Lehen dadoaren balioa 6 baldin bada, jokoa amaitzen da.
2. Bestela, lehem dadoa bikoitia baldin bada, bigarren dadaoren balioak esango du zein jokalarik mugitzen duen fitxa bat mahaira. Dadaoren balioa 1etik 6ra inoikoa denez, balio horri bat kenduko zaio [0, 5] tarteko balioa lortzeko. Jokalarriak fitxarik ez baleuka, ez da ezer egingo.
3. Lehen dadoa bakoitia baldin bada, fitxa bat mugituko da mahaitik bigarren dadaoren balioak esango duen jokalarira. Fitxa hori jokalarriak erabiliko duen azkena izango da. Mahaien fitxarik ez balego, ez da mugimendurik egingo.
4. Irabazlea fitxa beltz gehien duen jokalaria izango da (bankua kontuan hartu gabe). Berdinketaren kasuan, irabazlea puntuazio maximoa duen lehen jokalaria izango da.

Adibidez, hasierako egoeran dagoek (4, 3) emango balute, bigarren jokalarriaren fitxa bat hartuko litzateke (3 ken 1) eta mahaian jari beharko litzateke:

j0 (bankua)					
j1					
j2					
j3					
j4					
j5					

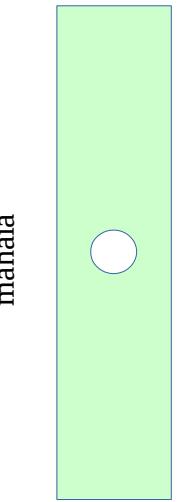
Dadoak berriro botatzen direnean (4, 1) bikotea aterako balitz, fitxa bat mugituko da Ogarren jokalaritik mahaira:

j0 (bankua)					
j1					
j2					
j3					
j4					
j5					



Eta orain (3, 5) aterako balitz, lehenengo dado bakoitia denez, fitxa bat mugituko da mahaitik 4garren jokalarira. Kontuan izan mugituko den fitxa mahaien kokatu zen azkenekoia izango dela. Gainera, fitxa hori 4garren jokalaria jokatuko duen azkena izango da.

j0 (bankua)					
j1					
j2					
j3					
j4					
j5					



Azpiprograma hau implementatu behar da:

```
public class Jokoa {
    Queue<Integer>[] jokalariak;
    // Fitxaen koloreak balio osokoien bidez adierazten dira: beltzak 0 eta
    // beste jokalarien kolorea bere posizioarekin bat etorriko da (hau da,
    // 1 jokalariak 1 kolorea, ...)

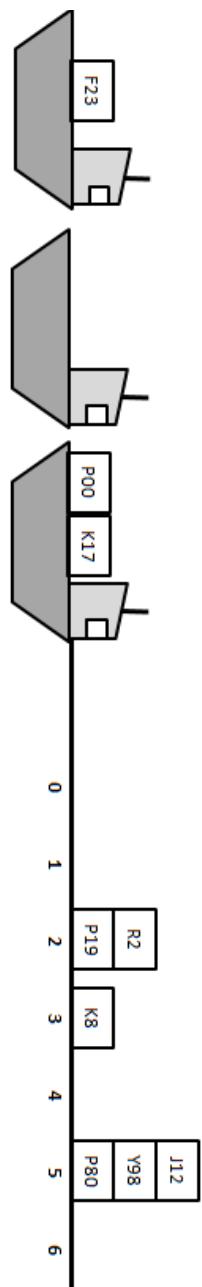
    Stack<Integer> mahaia;

    public int jokoa(int n, ArrayList<Jokaldi> jokaldiak) {
        // aurre: n jokalari bakoitzaren hasierako fitxa-kopurua da
        // "jokaldiak" zerrendak partida bateko jokaldiak ditu
        // post: emaitza irabazlearen zenbakia da
    }

    public class Jokaldi {
        int dado1;
        int dado2;
    }
}
```

2. ariketa: Portua (1,5 puntu)

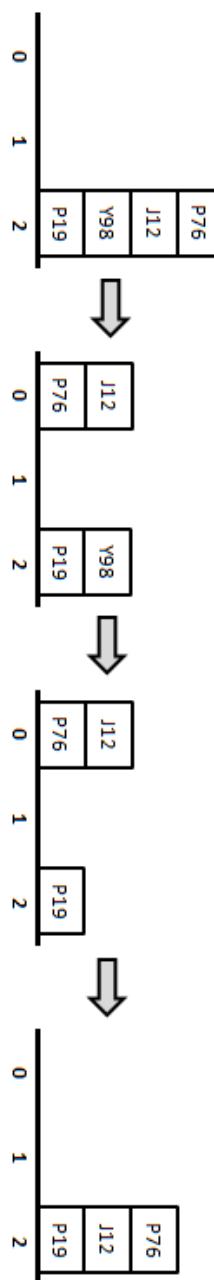
Portu bateko jardueraren simulazioa egin nahi dugu. Portuak hainbat nasa ditu kontainerrak pilatzeko. Eguna hasten denean, ontziak ilaran sartzen dira portura, nasetatik kontainerrak kargatzeko edo nasetan kontainerrak deskargatzeko.



Ontzi bakoitza mota batekoa da (kargako ontzia edo deskargako ontzia) eta hainbat eskaera ditu betetzeko. Eskaera bakoitzak kontainer baten kodea eta nasa baten zembakia ditu. Deskargako ontziek nasetan kontainerrak deskargatzeko eskaerak izango dituzte, eta kargako ontziek nasetatik kontainerrak kargatzekoak. Badakigu ere 0 zenbakia duen nasa berezia dela eta ez dela eskaeretan agertuko.

Horrela funtzionatuko du jardueraren simulazioak, ontzien ilaran ontziak dauden bitartean:

- ◆ Ilarako lehen ontzia hartuko da:
- ◆ Deskargako ontzi bat bada:
 - Ontziaren lehenengo *maxEskaera* eskaerak kudeatuko dira (gutxiago baditu, dituen guztiak):
 - Eskaera bakoitzeko: kontainerra adierazitako nasan utziko da.
 - *maxEskaera* eskaera baino gehiago bazituen, ontzia ontzien ilaran jarriko da berri, ilararen amaieran.
 - Kargako ontzi bat bada:
 - Ontziaren eskaera guztiak kudeatuko dira:
 - Eskaera bakoitzeko: kontainerra adierazitako nasa kendu behar da. Kontainer bat kendu behar denean, 0. nasa erabili dezakegu aldi baterako bere gainean dauden kontainerrak uzteko, eta behin gure kontainerra kendu dugula, 0. nasa utzi ditugunak jatorrizko nasan utziko ditugu berriz. Adibidez, ondorengo irudiiko Y98 kontainerra kentzeko:



Portua klasearen portuaSimulatu metodoa implementatzea eskatzen da.

```
public class Eskaera {
    String kontainerKode;
    int nasa;
}

public class Ontzia {
    int mota; //0 (deskarga), 1 (karga)
    String izena;
    Queue<Eskaera> eskaerak;
}

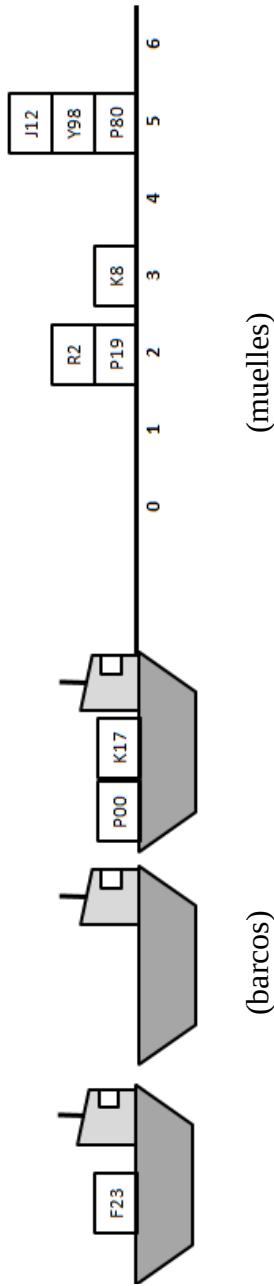
public class Portua {
    Stack<String> nasaak[];

    public void portuaSimulatu(Queue<Ontzia> ontziak, int maxEskaera, int nasaKop) {
        //Aurre: maxEskaera deskargako ontzi batek txanda battean kudeatu ditzakeen
        //eskaera kopurua da (>=1)
        //Aurre: nasaKop portuaren nasa kopurua da (>=2). 0. nasa berezia da.
        //Post: portuaren jarduera simulatu da, ontzien eskaerak betez.

        //Nasak sortu
        ...
        ...
        //Portuaren jarduera simulatu
        ...
    }
}
```

Ejercicio 2: Puerto (1,5 puntos)

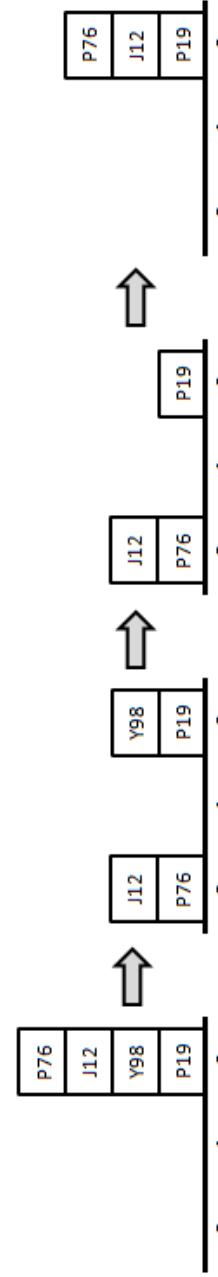
Queremos simular la actividad de un puerto. El puerto tiene diferentes muelles para guardar los contenedores que llegan. Cuando empieza el día, los barcos entran ordenadamente al puerto para cargar y descargar contenedores a y desde los muelles.



Cada barco es o bien de carga o de descarga, y tiene una serie de peticiones a completar. Cada petición tiene un código de contenedor y un número de muelle. Los barcos de descarga tendrán peticiones para descargar contenedores en el muelle, y los de carga para cargar contenedores desde el muelle. El muelle de código 0 es especial y no aparecerá en las peticiones.

La simulación de la actividad del puerto funciona de la siguiente manera, hasta que se atiendan todas las peticiones de los barcos:

- ◆ Se atenderá el primer barco que ha llegado.
- ◆ Si es un barco de descarga:
 - Se atenderán las primeras *maxPeticiones* de un barco (si tiene menos, todas)
 - Para cada petición: se dejará el contenedor en el muelle especificado.
 - Si el barco tuviera más de *maxPeticiones*, se colocará el barco al final de la cola con las peticiones restantes.
- ◆ Si es un barco de carga:
 - Se atenderán todas las peticiones del barco:
 - Para cada petición: el contenedor se cogerá del muelle especificado. Para sacar un contenedor, se puede usar el muelle 0 para dejar ahí temporalmente los contenedores que se encuentran encima del contenedor pedido y, una vez quitado ese contenedor, se volverán a colocar en ese muelle los contenedores apilados temporalmente en el muelle 0. Por ejemplo, en la siguiente figura se muestra cómo se cargaría en el barco el contenedor Y98:



Se pide implementar el método *simularPuerto* de la clase *Puerto*.

```

public class Peticion {
    String codigoDeContenedor;
    int muelle;
}

public class Barco {
    int tipo; // 0 (descarga), 1 (carga)
    String nombre;
    Queue<Peticion> peticiones;
}

public class Puerto {

    Stack<String> [] muelles;

    public void simularPuerto(Queue<Barco> barcos, int maxPeticiones, int numMuelles) {
        // Pre: maxPeticiones es el número máximo de peticiones que se pueden atender en
        //      el turno de un barco
        // Post: numMuelles es el numero de muelles del puerto (>=2). El muelle 0 es especial
        //       se ha simulado la actividad del puerto, atendiendo las peticiones de los barcos

        // crear los muelles
        ...
    }

    public void simularPuerto() {
        ...
    }
}

```

3. gaiak: Errekurtsibitatea

Bertol Arrieta eta Koldo Gojenolaren gardenkiak



Errekurtsibitatea

- Metodo (eragiketa) bat errekurtsiboa da, baldin eta bere buruari deitzen badio
- Diseinu errekurtsiboak oinarrizko kasuak eta kasu orokorrak hartu behar ditu kontuan

Adibidea: batukaria

DEA

$$\sum_{k=1}^N k = N + \sum_{k=1}^{N-1} k$$

```
public int suma (int num) {  
    int result;  
    if (num == 1)  
        result = 1;  
    else  
        result = num + suma (num-1);  
    return result;  
}
```

Errekurtsitatea

3

Errekurtsioa erabiltzeko klabeak

- Programa errekurtsibo batean derrigorrez egon behar da kasu ez errekurtsibo bat (gutxienez bat)
 - Kasu ez-errekurtsiboak (kasu nabariak): Emaitzia dei errekurtsiborik egin gabe lortzen dutenak
 - Kasu errekurtsiboek kasu nabarietara hurbiltzeko balio behar dute, hau da, programak bukatzera jo behar du.
- Kasu nabaririk ez duen programa errekurtsibo baten exekuzioa ez da inoiz amaituko

Errekurtsitatea

4

Azpiprogramma errekurtsiboen diseinurako pausoak

DEA

1. Espezifikazioa / parametrizazioa
2. Kasu nabarien (ez-errekurtsiboen) azterketa
3. Kasu errekurtsiboen (orokorren) azterketa
4. Algoritmoaren idazketa
5. Bukaeraaren azterketa
6. Implementazioa

Errekurtsibitatea

5

DEA

1. Espezifikazioa / parametrizazioa

- 1.1. Egin beharreko azpiprogramaren xehetasunak argitu
→ Aurrebaldintzak, postbaldintzak.
- 1.2. Azpiprogramaren parametroak eta parametro horien
mota finikatu (sarrera eta irteerakoak) → Kontuan izan dei
errekurtsiboa parametro horien arabera egingo dela.
- 1.3. Parametroek bete behar dituzten mugak eta murriketak
adierazi.
- 1.4. Hasierako deia definitu
→ bereziki garrantzitsua da parametroren batek
hasierako balio bat hartu behar badu.

Errekurtsibitatea

6

2. Kasu nabarien azterketa

1. Zehaztu zein diren kasu nabariak
(gutxienez kasu nabari bat !!)
2. Kasu nabari horietarako adierazi zein den eman beharreko emaitza

3. Kasu orokorren azterketa

1. Zehaztu zein diren kasu orokorrak: parametroen zein balioetarako egin behar diren dei errekurtsiboa.
2. Kasu orokoren tratamendua
 - Kasu nabarien eta orokoren artean gerta litezkeen kasu guztiak bildu behar dira.
 - **Dei errekurtsiboen parametro errealek kasu nabarietara hurbiltzeko balio behar dute.**
 - Dei errekurtsiboean erabiltzen diren parametroek bat etorri behar dute parametro formalekin (parametrizazioan definituakoak)
→ motan eta kopuruan

4. Bukaeraren azterketa

Egiaztatu dei errekurtsiboean parametroak kasu nabarietara hurbiltzen doazela, eta, beraz, beti bukaera iritsiko garela.

5. Algoritmoaren idazketa

- Algoritmoa idazten da kasu desberdin guztien definizioak bilduz, txukunduz eta, ahal bada, trinkotuz.
- Behar baldin bada, datu-egituraren diseinua ere pentsatu

6. Implementazioa

1. Programazio-lengoain idatzi algoritmoa
2. Datu-egituren xehetasunak finkatu
3. Eraginkortasunean irabazteko aldaketak egin

[Errekurtsibitatea](#)

11

Iterazioaren eskema

algoritmo Iterazioa

hasiera

Hartu_lehenengo_osagaia (Osag)

bitartean ez (Azkeneko_osagaia_da (Osag))
egin

Tratatu_osagaia (Osag)

Hartu_hurrengo_osagaia (Osag)

ambitartean

Tratatu_osagaia (Osag)

amaiia

[Errekurtsibitatea](#)

12

Errekurtsioaren eskema (1)

DEA

algoritmo Errekurtsioa1
hasiera
baldin bukaera-baldintza orduan
azkeneko pausuak burutu
bestela
ekintza orokorrak egin
hurbildu bukaera-baldintzara
deitu berriro prozesuari parametro berrieikin
ambaldin
amaia

[Errekurtsibitatea](#) 13

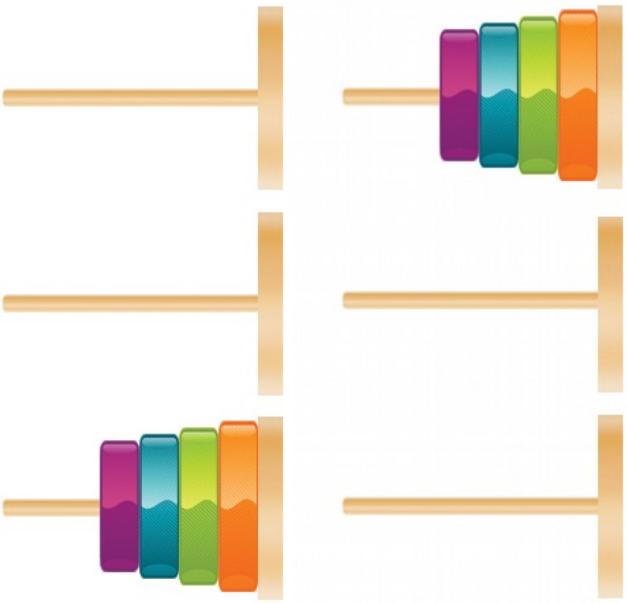
DEA

Errekurtsioaren eskema (2)

algoritmo Errekurtsioa2
hasiera
baldin ez (bukaera-baldintza) orduan
ekintza orokorrak egin
hurbildu bukaera-baldintzara
deitu berriro prozesuari parametro berrieikin
ambaldin
amaia

[Errekurtsibitatea](#) 14

Hanoiko dorreak



Errekurtsibitatea

15

Hanoiko dorreak

```
// k diskak mugituko dira x dorretik y dorrera  
// z dorrea laguntza moduan erabiliz.  
// mugitu(x,y) = "diska bat mugitu x dorretik y dorrera"
```

```
public static void hanoi(int k, int x, int y, int z){  
    if (k==1)  
        mueve(x,y);  
    else{  
        hanoi(k-1, x, z, y);  
        mueve(x,y);  
        hanoi(k-1, z, y, x);  
    }  
}
```

Errekurtsibitatea

16

Adibidez: Bilaketa dikotomikoa

(Bilaketa bitarra ere deitua)

- Array ordenatu bat izanik:


```
// taula[i..f] osokoen array ordenatua da (txikitik handira)
// x taula[i..f]n baldin badago, bere indizea itzultzen du.
// Bestela, -1 itzultzen du

public int bilaketaDikotomikoa(int i, int f, long x){
    if (i>f)
        return -1; //x ez dago taulan
    else{
        int erdia = (i+f)/2;
        if (taula[erdia]==x)
            return erdia;
        else if (taula[erdia]>x)
            return bilaketaDikotomikoa(i, erdia-1, x);
        else
            return bilaketaDikotomikoa(erdia+1, f, x);
    }
}
```

Errekurtsibitatea

17

Algoritmo errekurtsiboen analisia

- Kostu-funtzioaren kalkuluua:
 - Batukaria:
 - $f(n) = O(1) + f(n-1)$
 - Bilaketa dikotomikoa:
 - $h(n) = O(1) + h(n/2)$

Errekurtsibitatea

18

Irakurgaiak

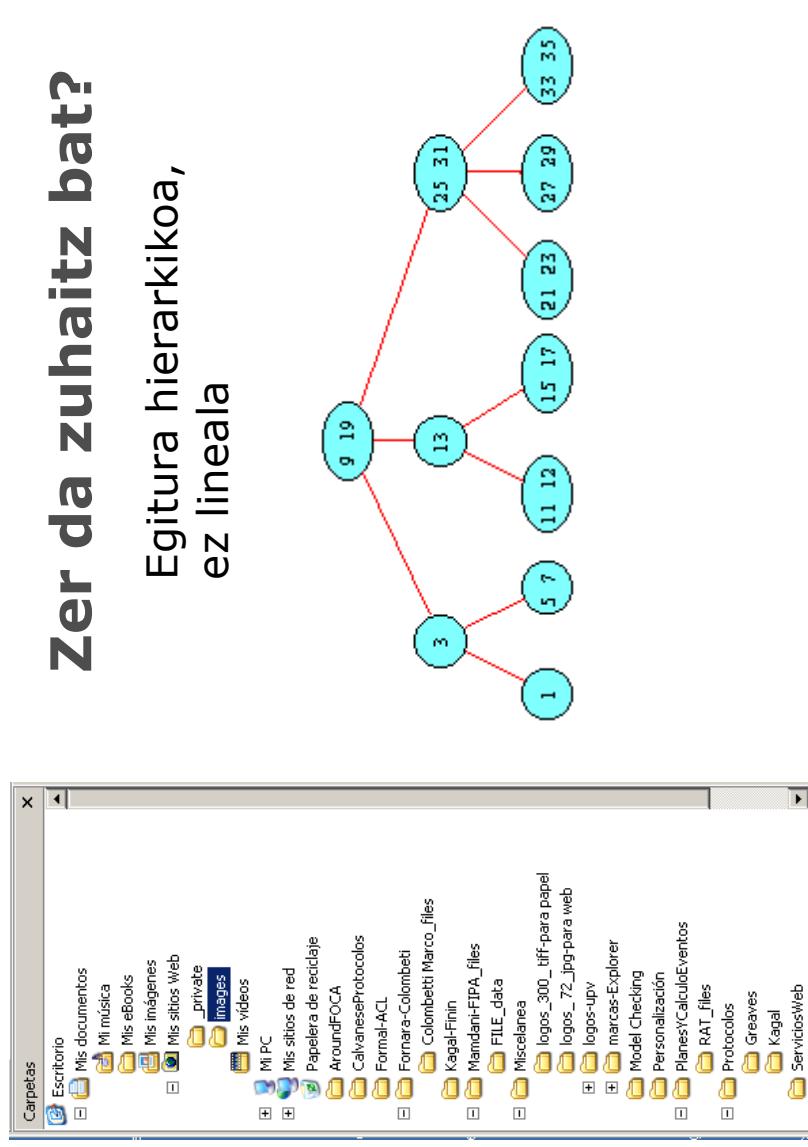
- [Lewis, Chase 2010]
 - 7. kapitula

Zuhaitzak

DEA

Zer da zuhaitz bat?

Egitura hierarkikoa,
ez lineala



Zuhaitzak

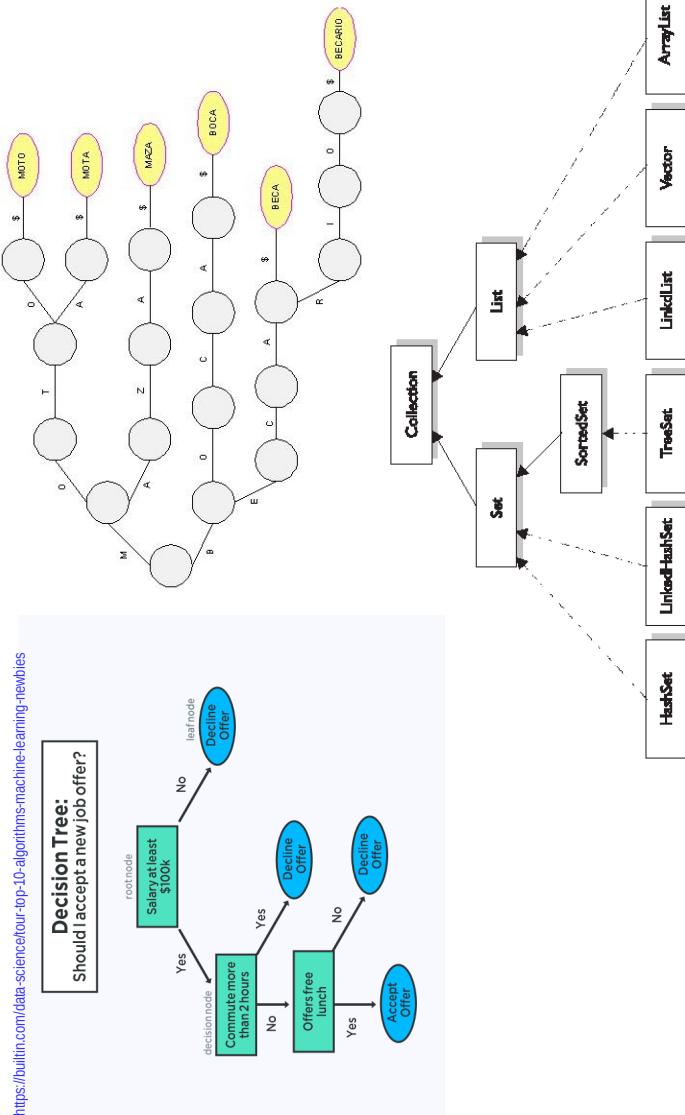
Oinarrizko terminologia

- **Zuhaitz** bat **adabegiez** eta **arkuez** osatuta dago. Arkuek adabegiak konektatzen dituzte
- Adabegiek entitateak adierazten dituzte, eta arkuek entitateen arteko erlazioak

Zuhaitzak

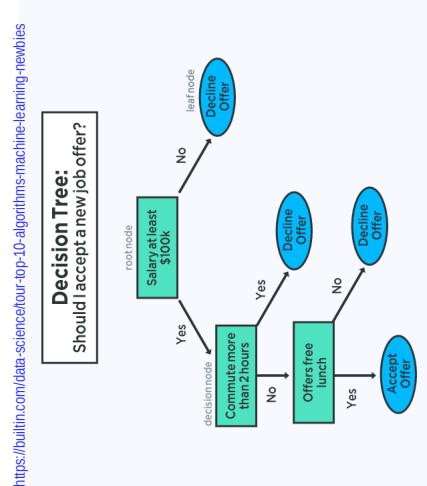
3

Adibideak



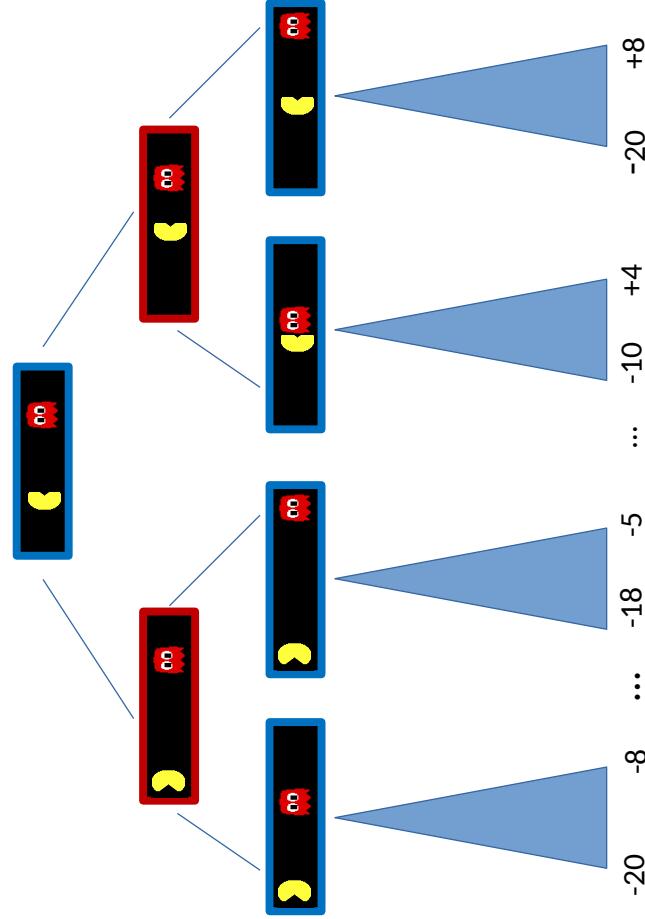
Zuhaitzak

4



Adibideak

DEA



Zuhaitzaren kontzeptua

DEA

- Zuhaitz bat arkuek konektatutako adabegiez osatuta dago:
 - Erro-adabegi bakarra dago
 - Adabegi balkitzak, erroak ezezik, guraso bakarra du
- Ondorioa: zuhaitz batean bide bakarra dago errotik edozein adabegiraino

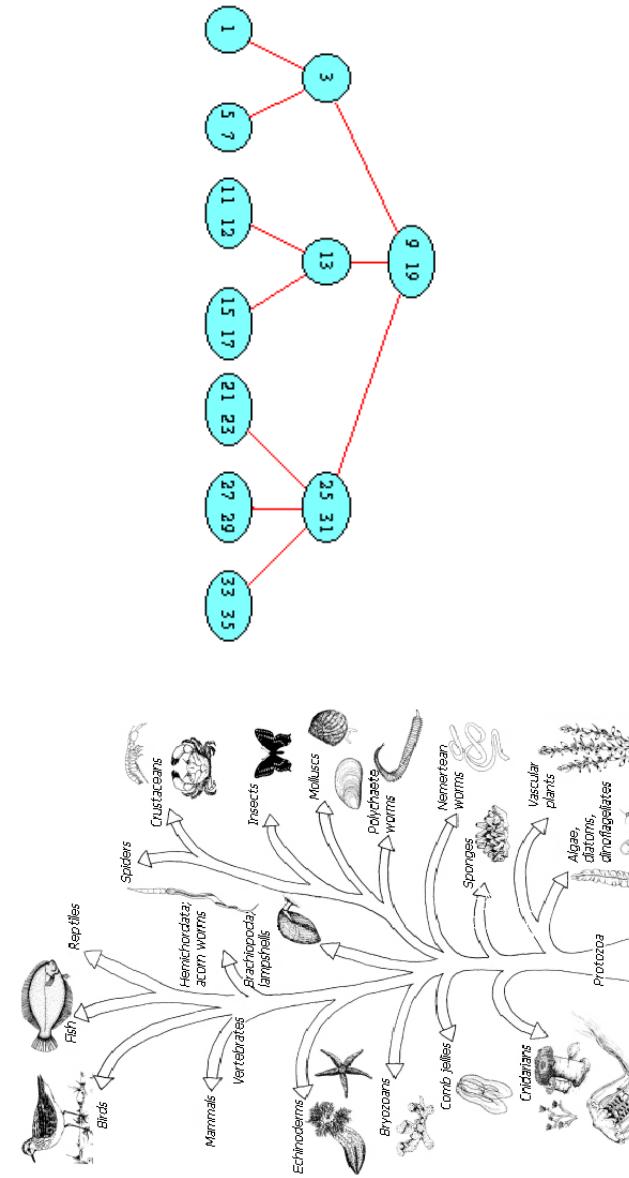
Zuhaitzaren definizio errekurtsiboa

- T motako zuhaitzen definizioa:
 - $< >$ \in Zuhaitz $< T >$ [zuhaitz hutsa, edo bestela]
 - $A_1, A_2, \dots, A_n \in$ Zuhaitz $< T >$ eta $t \in T \rightarrow t (A_1, A_2, \dots, A_n) \in$ Zuhaitz $< T >$
- [T motako objektu bat (erroa) eta T motako zuhaitzen kopuru finitua]

Zuhaitzak

7

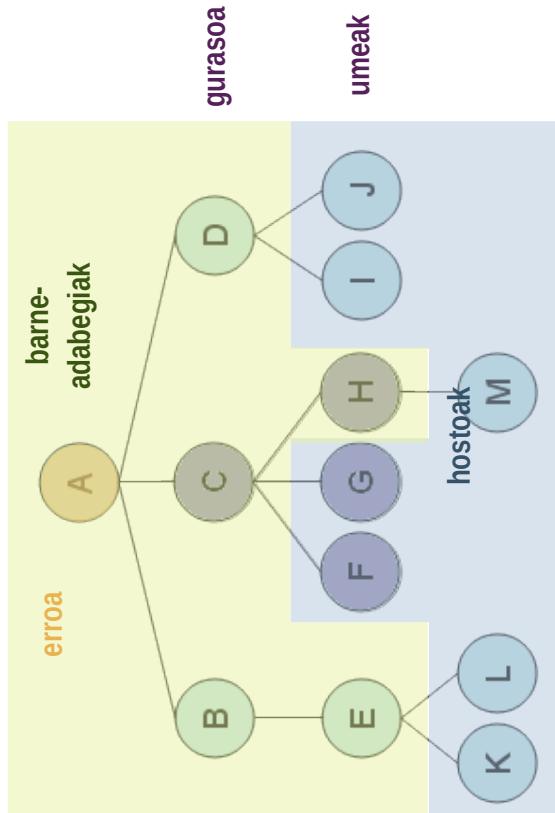
Zuhaitzen diagramak



Zuhaitzak

8

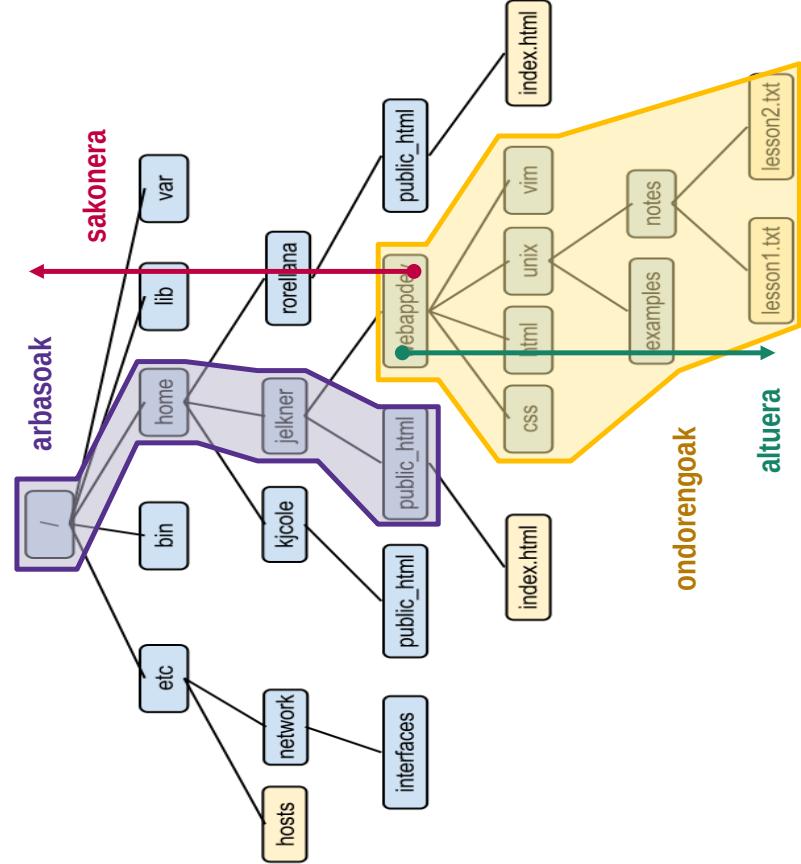
Terminologia eta kontzeptuak



09/06/21

Datu-Egiturak eta Algoritmoak

9

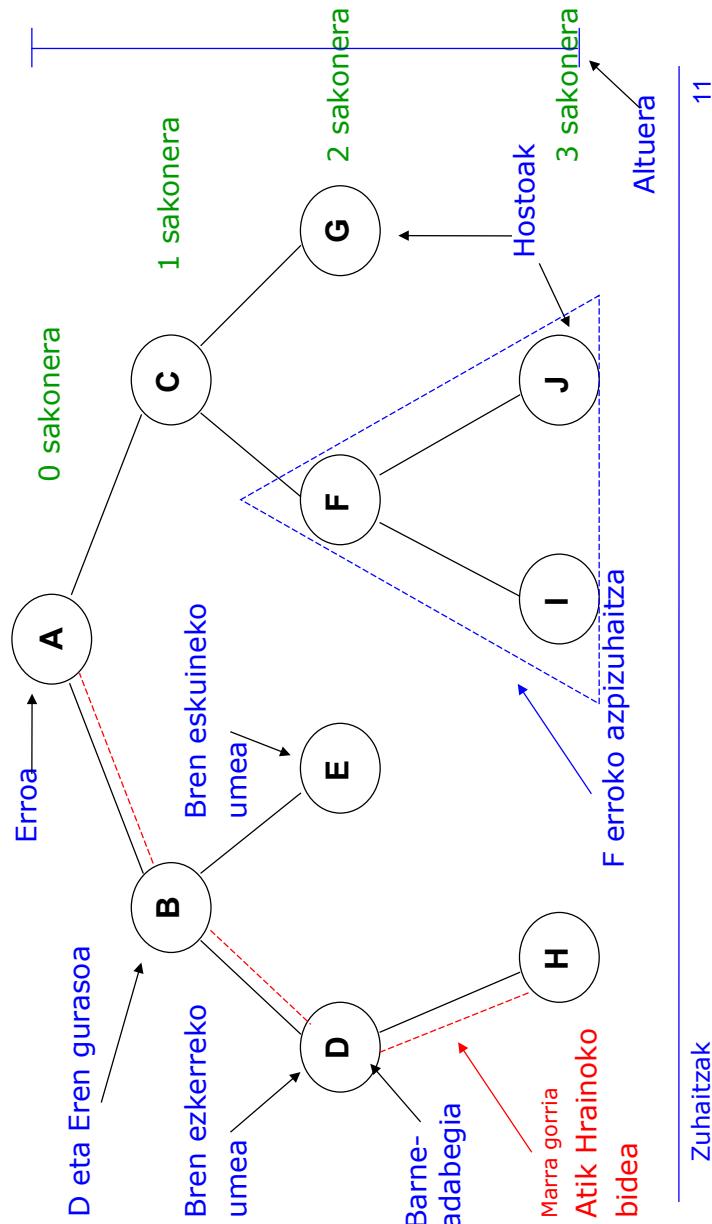


09/06/21

Datu-Egiturak eta Algoritmoak

10

Oinarrizko terminologia



Oinarrizko terminologia

- Adabegi baten **altuera** adabegi horretatik hosto batera dagoen bide luzeenaren tamaina da
- **Hosto-adabegi baten altuera** zero da
- **Zuhaitz baten altuera** erroaren altuera da
- Adabegi baten **sakonera** errotik adabegi horretara doan bidearen luzera da
- Adabegi baten **urrekoak**, adabegia bera eta bere gurasoarenurreko guztiaik dira
- Adabegi baten **ondorengoa**, adabegia bera eta bere umeen ondorengoaik dira

Oinarrizko terminologia

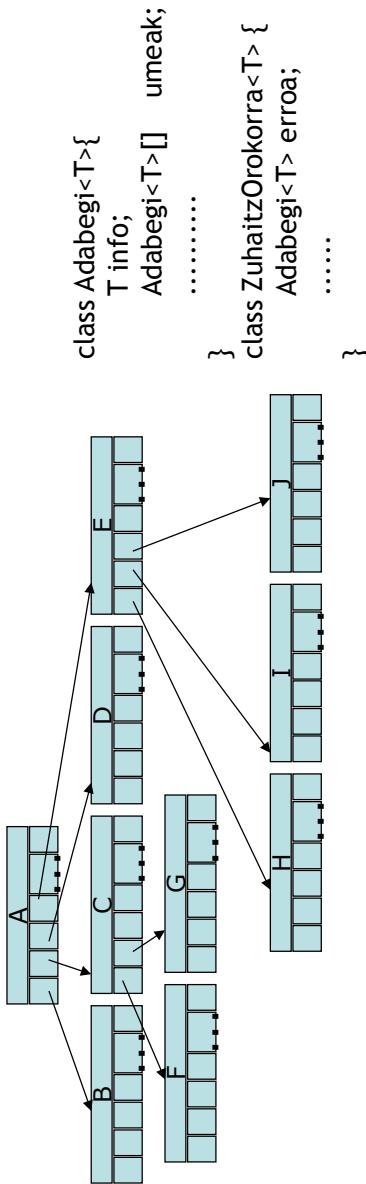
- Adabegi baten **gradua** adabegi horren ume-kopurua da
- **Zuhaitz baten gradua** bere adabegien gradu handiena da
 - Zuhaitz n-tarra: n graduko zuhaitza
 - Zuhaitz bitarra: 2 graduko zuhaitza

Zuhaitzen adierazpena eta implementazioa

- Zuhaitzen adierazpena eta implementatutako metodoak zuhaitz horien erabileraen araberakoak izango dira

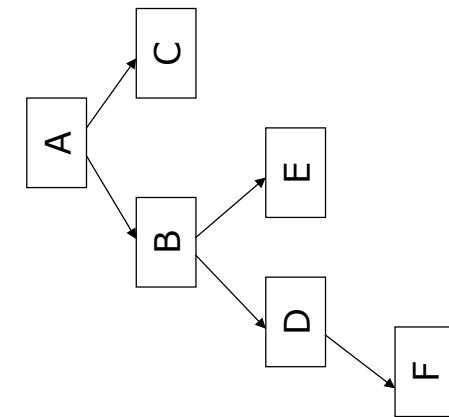
Zuhaitz orokorrak implementatzeko estrategiak (1)

- Adabegiak erreferentzien arrayak
- Memoria alferrik erabiliko da ume-kopurua oso aldakorra bada
- Arazoak arrayaren luzerarekin



Zuhaitzak 15

Zuhaitz orokorrak implementatzeko estrategiak (2)



	0	1	2	3	4	5
A	C	B	E	D	F	
2	1		3		5	

Arrayaren bidezko adierazpena, estekak simulatuz

Zuhaitzak 16

Zuhaitz orokorrak implementatzeko estrategiak (3)

- Umeak zerrenda estekatu batean

Diagram illustrating the execution flow of a linked list insertion operation. The diagram shows a sequence of nodes A through J, each containing a value and a pointer to the next node. Nodes A through G are part of one list, while H through J are part of another. The pointer from G points to the head of the second list. The code shows the insertion of node I into the second list, with pointers being updated to reflect the new structure.

```

class ZuhaitzOrokorra<T>{
    AdabegiZO<T> erroa;
    ...
}

class AdabegiZO<T>{
    T info;
    LinkedList<T> umeak;
    ...
}

class LinkedList<T>{
    Adabegi<T> hasiera;
    ...
}

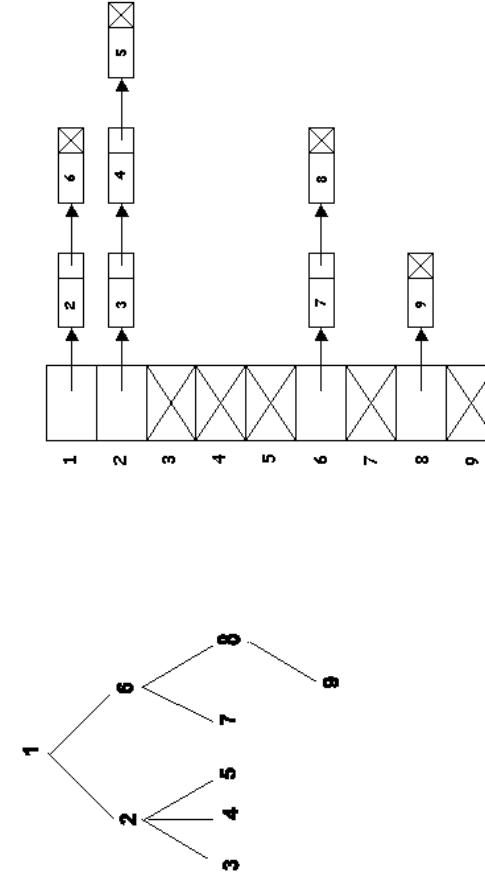
class Adabegi<T>{
    AdabegiZO<T> data;
    Adabegi<T> hurrengoa;
    ...
}

```

Zuhaitzak

17

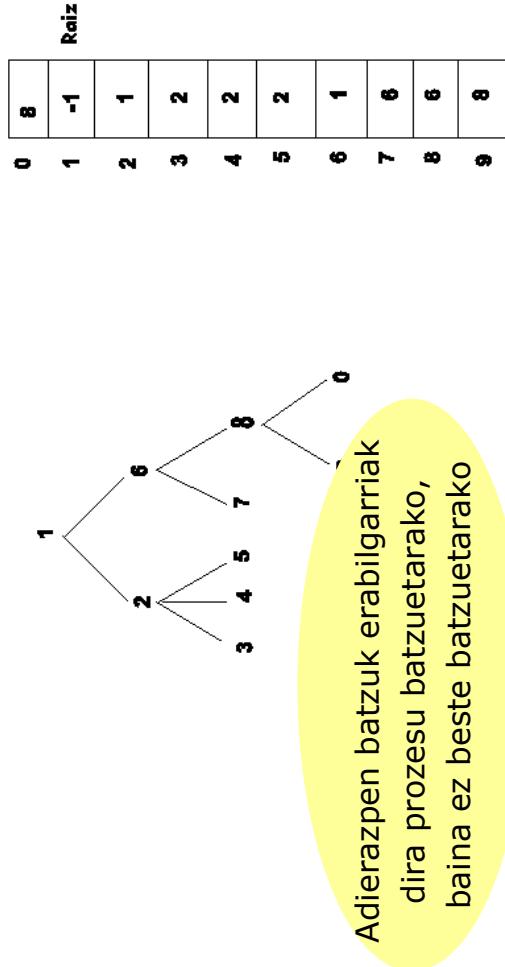
Zuhaitz orokorrak implementatzeko estrategiak (4)



Umeen zerrenden hizetako adierazmena

Zentral

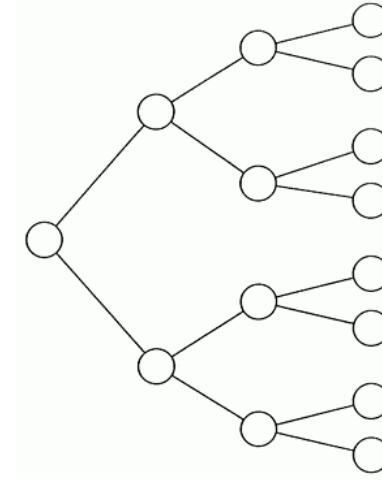
Zuhaitz orokorrak implementatzeko estrategiak (5)



Zuhaitzak 19

Zuhaitz bitarra

- Adabegi bakoitzak, gehienez 2 ume ditu (ezkerreko eta eskuineko umea)



Zuhaitzak 20

Zuhaitz bitarrak implementatzeko

DEA

- Adabegi bakotitzak:

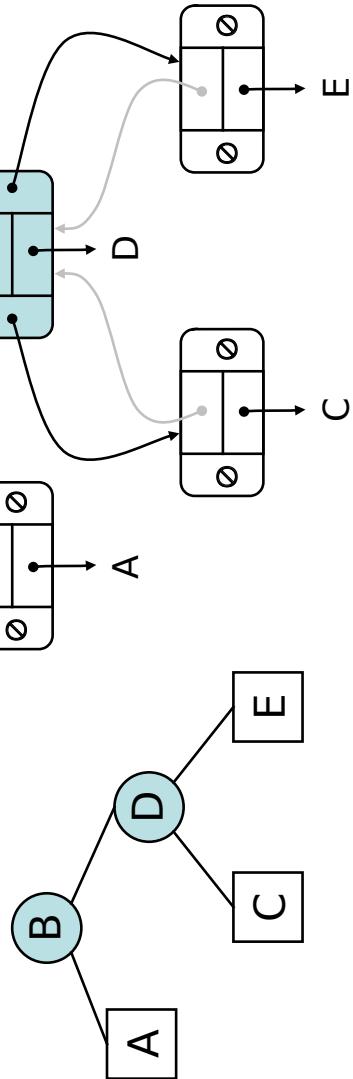
- Elementua

- Ezkerreko umearen adabegia

- Eskuineko umearen adabegia

- [Adabegi gurasoa]

- Zuhaitza: erro-adabegiaren erreferentzia



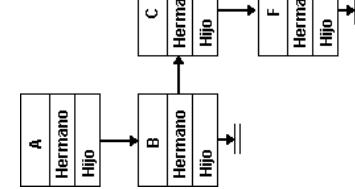
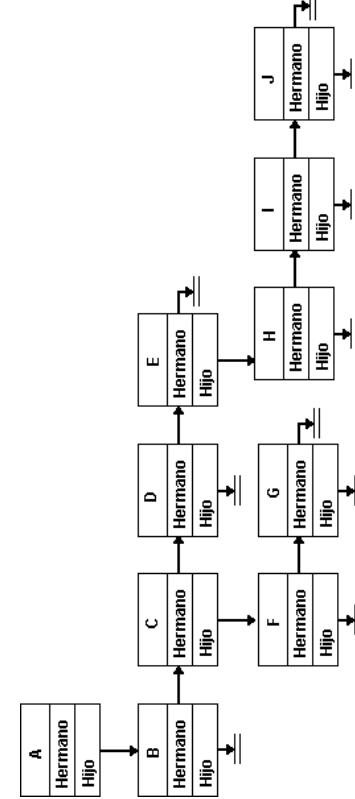
Zuhaitzak

21

Edozein zuhaitz implementa daiteke zuhaitz bitar baten bidez

DEA

- Erreferentzia bat erlazio bakoitzeko (umea/anaia)



```
class Adabegi<T> {  
    T info;  
    Adabegi<T> umea;  
    Adabegi<T> anaia;  
    ...  
}  
class ZuhaitzOrokorra<T> {  
    Adabegi<T> erroa;  
    ...  
}
```

Zuhaitzak

22

Gure zuhaitz bitarrentzako interfazea

```

public interface ADTBinaryTree<T> {

    // ADABEGIEN ATZIPENERAKO
    public boolean isEmpty();                                // KONTSULTARAKO
    // equals(elem) duen balioa topatzeko                      // Zuhaitzaren adabegi-kopurua bueltatuko du
    /*.equals(elem) duen balioa topatzeko du
     * (Hau da, T motan equals metodoaren
     * implementazioaren arabera funtzionatuko du,
     * eta objektuaren erreferentzia bueltatuko du,
     * topatzeko badu null bueltatuko du ez baldin badago
     */
    public T find(T elem);                                 // true bueltatuko du zuhaitzak balio hori badu,
    // eta false bestela
    public boolean contains(T elem);

    public String toString();                             // ALDAKETARAKO

    // ezkerreko azpizuhaitza kentzen du
    public void removeLeftSubTree();                     // ITERADOREAK
    // eskuineko azpizuhaitza kentzen du
    public void removeRightSubTree();                   // ITERADOREAK
    // adabegi guztiak kentzen ditu
    public void removeAll();                            // ITERADOREAK
}

```

Zuhaitzak

25

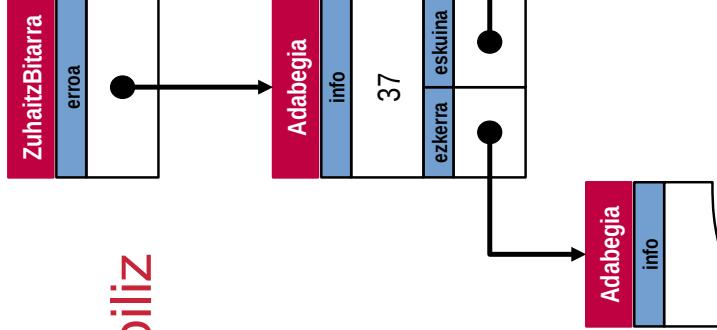
Zuhaitz bitarren implementazioa, memoria dinamikoa erabiliz

- **Adabegiak errekurtsiboki definitzen dira:**

- Ezkerreko eta eskuineko umeaek adabegien erreferentziak dira

- **ZuhaitzBitarra** eta **Adabegia** bereizi egin behar dira

- **ZuhaitzBitarra**k adabegi erroaren erreferentzia gordetzen du

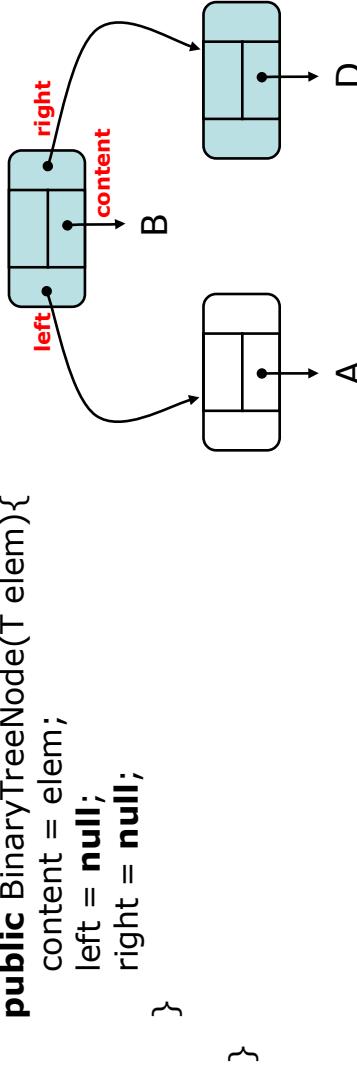


Adabegien klasea zuhaitz bitarretan

```
public class BinaryTreeNode<T> {

    protected T content;
    protected BinaryTreeNode<T> left;
    protected BinaryTreeNode<T> right;

    public BinaryTreeNode(T elem){
        content = elem;
        left = null;
        right = null;
    }
}
```



Zuhaitzak

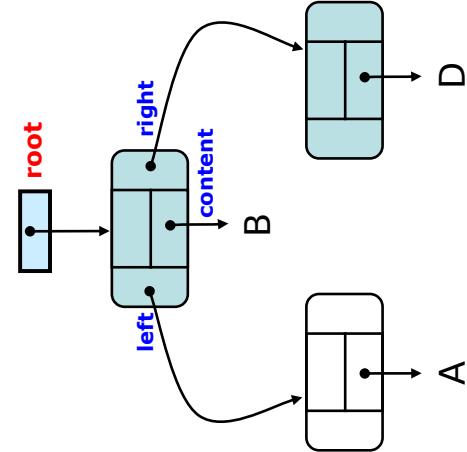
27

Adabegien klasea zuhaitz bitarretan

```
public class BinTree<T> implements ADTBinaryTree<T> {

    protected BinaryTreeNode<T> root;
    protected int count;

    public BinTree(){
        root = null;
        count = 0;
    }
}
```



Zuhaitzak

28

Zuhaitzak eta errekurtsibitatea

```
// ezkerreko azpizuhaitza kentzen du
public void removeLeftSubTree(){
    count = count - numDescendants(root.left);
    root.left = null;
}

// "adabegia"ren ondorengoa kalkulatzeko
private int numDescendants(BinaryTreeNode<T> adabegia){
    if (adabegia == null)
        return 0;
    else
        return 1 + numDescendants(adabegia.left) +
            numDescendants(adabegia.right);
}
```

Zuhaitzak

29

Errekurtsibitatea eta murgilketa

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

private boolean contains(T elem, BinaryTreeNode<T> temp){
    if (temp==null) // elem ez dago
        return false;
    else if (temp.content.equals(elem)) // elem temp-en dago
        return true;
    else
        return (contains(elem, temp.left) || contains(elem, temp.right));
}
```

contains(elem) murgilduta dago hemen: contains(elem, root)

Zuhaitzak

30

Zuhaitz bitarretako elementu guztien atzipena

DEA

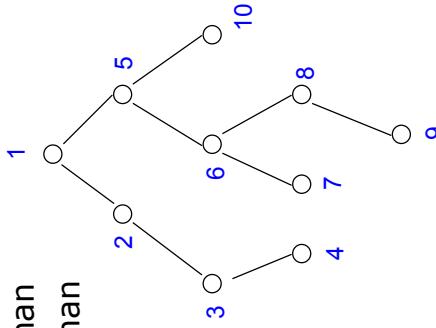
Eragiketa	Deskribapena
Iterator<T> iteradoreInOrdena()	In-Ordena azterketarako iteradorea bueltatzen du
Iterator<T> iteradoreAurreOrdena()	Aurre-Ordena azterketarako iteradorea bueltatzen du
Iterator<T> iteradorePostOrdena()	Post-Ordena azterketarako iteradorea bueltatzen du
Iterator<T> iteradoreMailaka()	Mailakako azterketarako iteradorea bueltatzen du

Zuhaitzak 31

DEA

Aurre-Ordena

1. Erroa bisitatu
2. Ezkerreko azpizuhaitza bisitatu aurre-ordenan
3. Eskuineko azpizuhaitza bisitatu aurre-ordenan

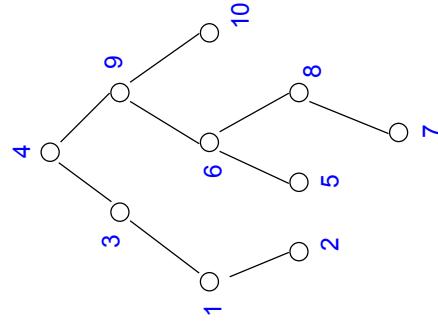


Zuhaitzak

32

In-Ordena

1. Ezkerreko azpizuhaitza bisitatu in-ordenan
2. Erroa *bisitatu*
3. Eskuineko azpizuhaitza bisitatu in-ordenan

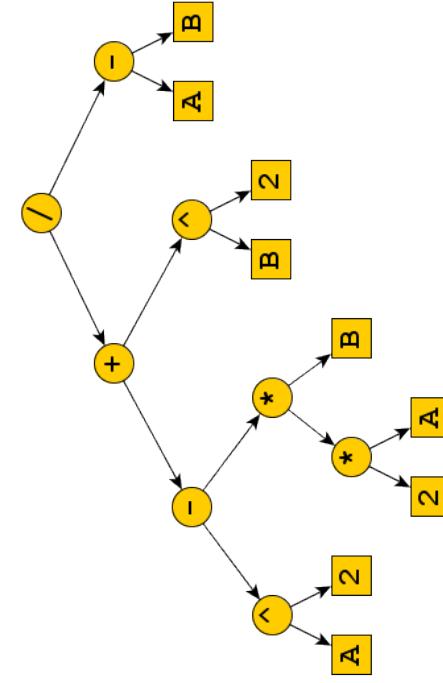


Zuhaitzak

33

Post-Ordena

1. Ezkerreko azpizuhaitza bisitatu post-ordenan
2. Eskuineko azpizuhaitza bisitatu post-ordenan
3. Erroa *bisitatu*

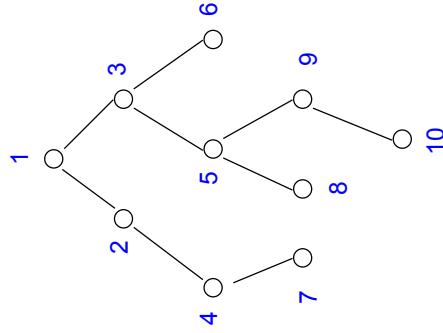


Zuhaitzak

34

Mailakako korritzea

- Sakonera bakoitzeko ($k=0$; $k++$):
ezkerretik eskuinera korritu k sakonerako adabegiak



Zuhaitzak

35

Post-Ordenaren bidezko iteradorea

```

public Iterator<T> iteradorPostOrdena(){
    LinkedList<T> tempList = new LinkedList<T>();
    postOrdena(root, tempList);
    return tempList.iterator();
}
  
```

DEA

35

```

private void postOrdena(BinaryTreeNode<T> erroa, LinkedList<T> lista){
    if (erroa != null ){
        postOrdena(erroa.left, lista);
        postOrdena(erroa.right, lista);
        lista.insertLast(erroa.content); // Adabegiaren prozesaketa
    }
}
  
```

Zuhaitzak

36

Mailakako iteradorea

```

public Iterator<T> iteradoreMailaka(){
    LinkedList<T> tempList = new LinkedList<T>();
    mailaka(root, tempList);
    return tempList.iterator();
}

private void mailaka(BinaryTreeNode<T> erroa, LinkedList<T> lista){
    if (erroa != null ){
        LinkdQueue<BinaryTreeNode<T>> ilara = new LinkdQueue<BinaryTreeNode<T>>();
        ilara.insert(erroa);
        while ( !ilara.isEmpty() ){
            BinaryTreeNode<T> temp = ilara.remove();
            lista.insertLast(temp.content); // Adabegiaren prozesaketa
            if (temp.left != null )
                ilara.insert(temp.left);
            if (temp.right != null )
                ilara.insert(temp.right);
        }
    }
}

```

Zuhaitzak

37

Irakurketa

[Lewis, Chase 2010]

-9. kapitula

Zuhaitza(informatika):

http://es.wikipedia.org/wiki/Árbol_binario

Zuhaitz bitarra:

http://es.wikipedia.org/wiki/Árbol_binario_de_búsqueda

Zuhaitzak

38

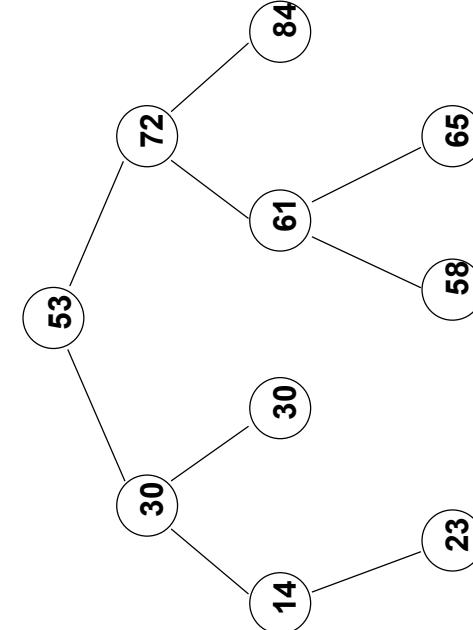
DEA

Bilaketa-zuhaitz bitarrak

DEA

Bilaketa-zuhaitz bitarra

- Zuhaitz bitarra da.
Adabegi bakoitzean, ezkerreko azpizuhaitzeko elementu guztiak erroko balioa baino txikiagoak dira, eta eskuneko 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 horri bueltatzen du
void removeAllOccurrences(T elem)	Elementu horren agerpen guztia ezabatuko ditu
T removeMin()	Zuhaitzeko elementu minimoa bueltatuko du
T removeMax()	Zuhaitzeko elementu handiena bueltatuko du
T findMin()	Zuhaitzeko elementu minimoa bueltatuko du
T findMax()	Zuhaitzeko elementu maximoa bueltatuko du

Bilaketa-zuhaitz bitarrak

3

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 guztia kenduko ditu
    public void removeAllOccurrences (T elem);

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

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

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

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

Comparable interfazea
implementatu behar du

Bilaketa-zuhaitz bitarrentzako klasea

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

Bilaketa-zuhaitz bitarrak

5

Bilaketa-zuhaitz bitarrentzako klasea

```
public class LinkedBinSearchTree<T> extends BinTree<T>
implements TADBinarySearchTree<T>{
    // protected BinaryTreeNode<T> root;
    // protected int count;
    /** Creates an empty binary search tree. */
    public LinkedBinSearchTree()
    {
        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 LinkedBinSearchTree (T element)
    {
        super (element);
    }
}
```

Comparable interface
implementatu behar du

DEA

Bilaketa-zuhaitz bitarrak

6

void addElement(T elem)

```

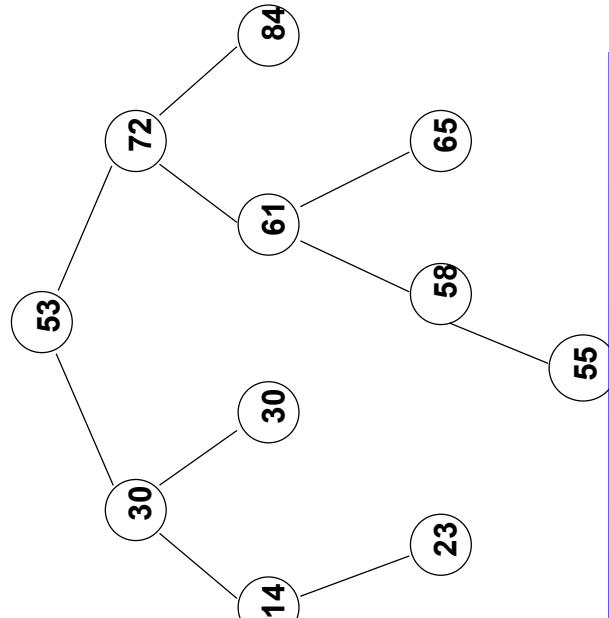
public void addElement(T element) {
    BinaryTreeNode<T> temp = new BinaryTreeNode<T>(element);
    Comparable<T> comparableElement = (Comparable<T>)element;

    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;
    }
}
    
```

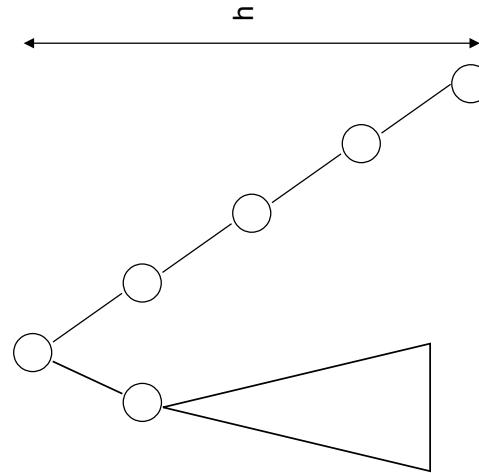
Bilaketa-zuhaitz bitarrak

7



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?



Bilaketa-zuhaitz bitarrak

8

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, implementazio errekurtsibo batek
 azpizuhaitzaren errora pasa behar du parametroen artean
```

Bilaketa-zuhaitz bitarrak

9

## T find(T elem), murgilketarekin

```
public T find(T elem) {
 Comparable<T> komp;
 return find(elem, root);
}

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

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

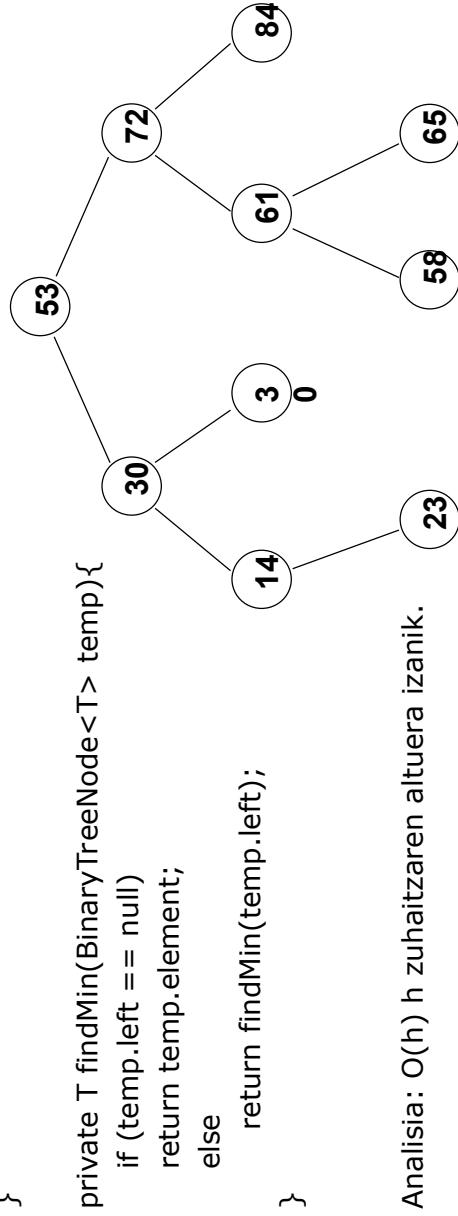
Bilaketa-zuhaitz bitarrak

10

## 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);
}
```



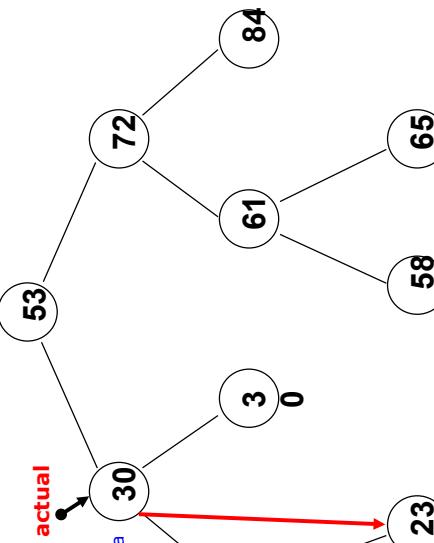
Analisia: O(h) h zuhaitzaren altuera izanik.

Bilaketa-zuhaitz bitarrak

11

## 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 eskueinoko azpizuhaitza: erroa minimoa da
 T temp = actual.right;
 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.

Bilaketa-zuhaitz bitarrak

12

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

Bilaketa-zuhaitz bitarrak

13

## 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);
}
```

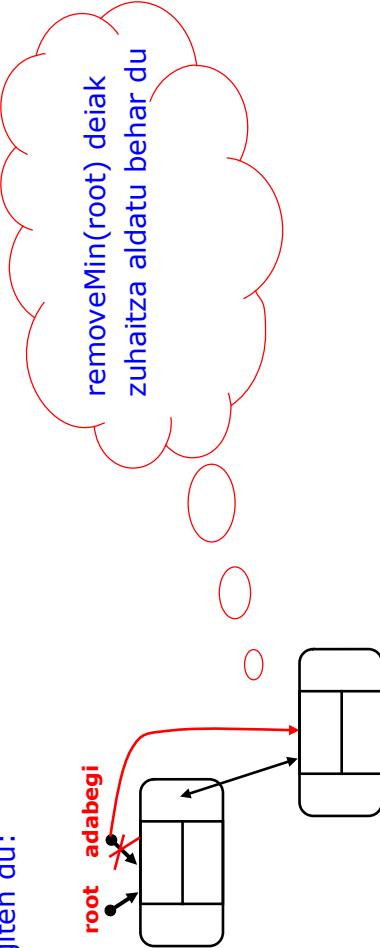
Hau da bueltatu behar den balioa, baina zuhaitzetik kendu behar da.  
Era adabegi.right ejiten badugu?  
EZ, horrela parametro formalaren balioa aldatzen da (aldagai lokala betatikoa da), baina ez du aldatzen parametro errealauren balioa

Bilaketa-zuhaitz bitarrak

14

## Parametroak aldagai lokalak bezalakoak dira

```
removeMin(root);
Demagun:
private T removeMin(BinaryTreeNode<T> adabegi){
 ... adabegi = adabegi.right; }
Hau egiten du:
```



Bilaketa-zuhaitz bitarrak

15

## T removeMin(), bertsio errekurtsiboa murgilketarekin

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

removeMin(adabegia)

“adabegia” erroa duen zuhaitzean minimoa ezabatutakoan geratzzen 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;
 }
}
```

Bilaketa-zuhaitz bitarrak

16

# Emaitzza removeMin(adabegia)

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

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

■ Analiisiai: O( $n \log n$ ) h<sub>k</sub> zuhaitzaren altuera izanik

17

# T removeElement(T elem).

## Kasuuen 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
    - Eskuiñeko azpizuhaitza badu, eta ez dago ezkerreko azpizuhaitzik, orduan erroa kendu eta bere balioa bueltatu, eskuiñeko azpizuhaitza utziz
    - Ezkerreko azpizuhaitza badu, eta ez dago eskuiñeko azpizuhaitzik, orduan erroa kendu eta bere balioa bueltatu, ezkerreko azpizuhaitza utziz
    - Bi azpizuhaitzak baldin badaude, kendu minimoa eskuiñeko 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)
  - eskuiñeko azpizuhaitzaean

# T removeElement(T elem)

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

DEA

Bilaketa-zuhaitz bitarrak

19

# Emaitzia removeElement(elem, adabegia)

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

Bilaketa-zuhaitz bitarrak

20

DEA

```

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

```

- Analisia: O(h) h zuhitzaren altuera izanik

## T removeElement(T elem) Bertsio errepikakorra (1)

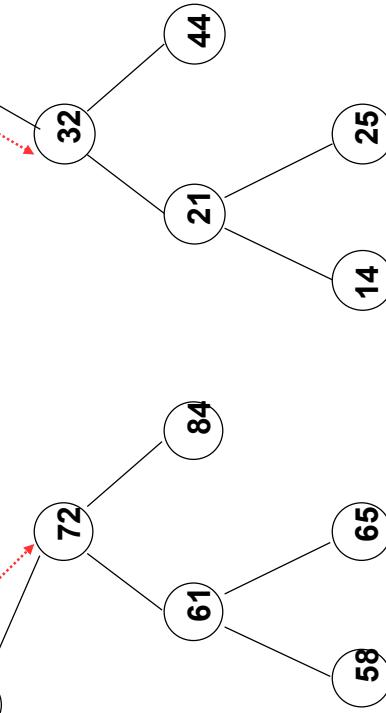
```
public T removeElement(T elem) {
```

Bilatu "elem" (adabegiaren erreferentzia (unekoia) eta gurasoarena gordeta)  
if ez aurkitua null bueltatu  
else // **ezabatu elementua**  
if elem = root then ezabatu root  
else ezabatu (unekoia, gurasoa)

DEA

## T removeElement(T elem) Bertsio errepikakorra (2)

ezabatu root:



BZB

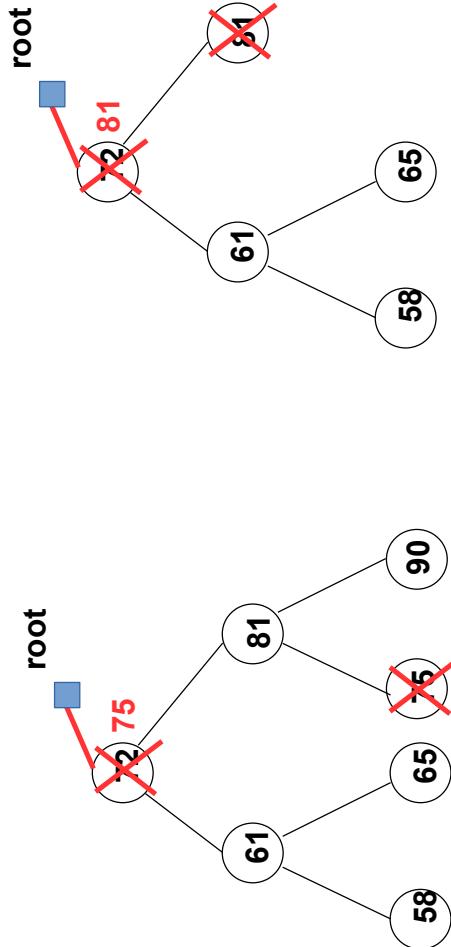
23

DEA

## T removeElement(T elem) Bertsio errepikakorra (3)

Ezabatu root, eskuin eta ezkerreko azpizuhaitzak ditu:

```
T berria = ezabatu txikiena root.right zuhaitzetik
root.content = berria;
```



BZB

24

## T removeElement(T elem) Bertsio errepikakorra (4)

ezabatu (unekoa, gurasoa), "uneko" hostoa da:

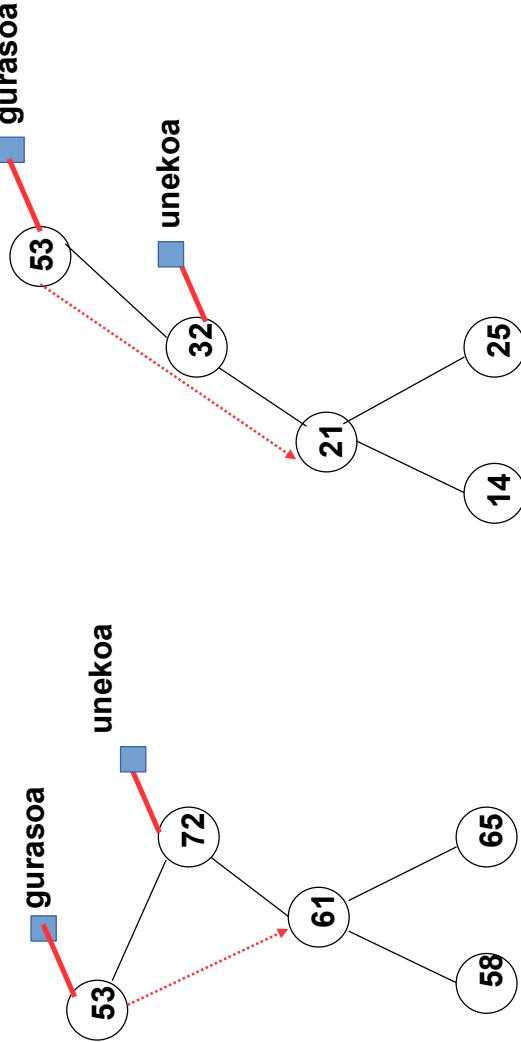


Kontuz: "uneko" ezkerreko edo eskuineko umea izan daiteke

BZB 25

## T removeElement(T elem) Bertsio errepikakorra (5)

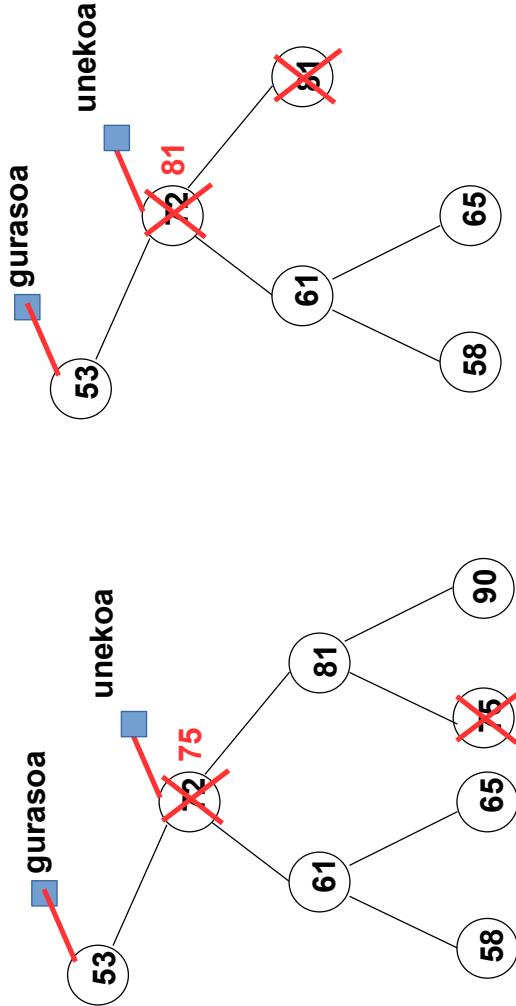
ezabatu (uneko, gurasoa), "uneko"-k bakarrik ezkerreko azpizuhaitza du  
(idem bakarrik eskuineko):



BZB 26

**T removeElement(T elem)**  
**Bertsio errepikakorra (6)**

ezabatu (unekoa, gurasoa), "uneko"-k ezkerreko eta eskuineko azpiuzhaitzak ditu



BZB 27

**T removeElement(T elem)**  
**Bertsio errepiakorra (7)**

```

public T removeElement(T elem) {
 BinaryTreeNode<T> unekoa = root;
 BinaryTreeNode<T> gurasoa = null;
 Boolean aurk = false;
 while (unekoa != null) && !aurk {
 if (unekoa.content.compareTo(elem) == 0) { aurk = true; }
 else if (unekoa.content.compareTo(elem) < 0) {
 gurasoa = unekoa;
 unekoa = unekoa.left;
 }
 else {
 gurasoa = unekoa;
 unekoa = unekoa.right;
 }
 }
}
} // while
If !aurk { return null; }
 \langle else { // ezabatu elementua

```

BZB 28

**T removeElement(T elem)**  
**Bertsio errepikakorra (8)**

```

public T removeElement(T elem) { // jarraitzen du
 else { // ezabatu elementua
 if (unecko == root) {
 if (root.left == null) && (root.right == null) { root = null;} // hostoa da
 else if (root.left == null) { root = root.right;}
 else if (root.right == null) { root = root.left;}
 else { // adabegiak bi ume ditu
 T balioBerria = removeMinBis(root.right);
 root.content = balioBerria;
 }
 }
 else { // adabegiko balioa ezabatu (gurasoia du)
 if (unecko.left == null) && (unecko.right == null) { // hostoa da
 if (unecko == gurasoia.right) { gurasoia.right = null;} else { gurasoia.left = null;}
 }
 else if (unecko.left == null)
 if (unecko == gurasoia.right) gurasoia.right = unecko.right;
 else gurasoia.left = unecko.right;
 else if (unecko.right == null)
 if (unecko == gurasoia.right) gurasoia.right = unecko.left;
 else gurasoia.left = unecko.left;
 else { // adabegiak 2 ume ditu
 T balioBerria = removeMinBis(unecko, unecko.right);
 unecko.content = halloBerria;
 }
 }
 }
}

```

29 BZB

```
void addElement(T elem)
Kasuen analisia
```

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

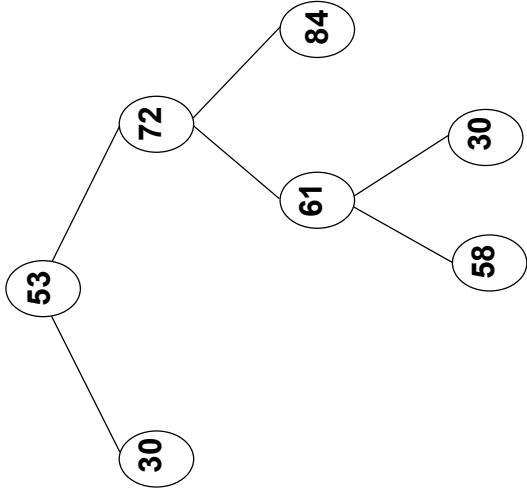
- Oinarritzko kasua:
  - Zuhaitz hutsa: elem erroan duen zuhaitza sortu
- Kasu orokorra:
  - Erroaren balioa > elem: gehitu ezkerreko azpizuhaitzean
  - Erroaren balioa < = elem: gehitu eskuineko azpizuhaitzean

## Zuhaitz bitarren motak

### Zuhaitz bitarren motak

- Zuhaitz bitar bat **betea** (*full*) da baldin eta soilik baldin barne-adabegi guztiek bi ume badituzte.
- Zuhaitz bitar bat **perfektua** (*perfect*) da baldin eta soilik baldin, betea izateaz gain, hosto guztiak maila berean badaude.
- Zuhaitz bitar bat **osoa** (*complete*) da baldin eta soilik baldin maila guztiak beteta badaude azken ezik; azkeneko mailan hosto guztiak ezkererra lerratuta egon behar dute.
- Zuhaitz bitar bat **orekatua** da baldin eta soilik baldin adabegi guztien azpiarbolean sakonerek gehienez ere bateko aldea badute.

## Zuhaitz bitar betea. Adibidea

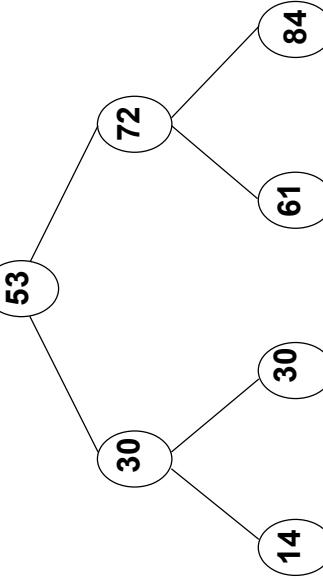


09/06/21

Datu-Egiturak eta Algoritmoak

33

## Zuhaitz bitar perfektua. Adibidea

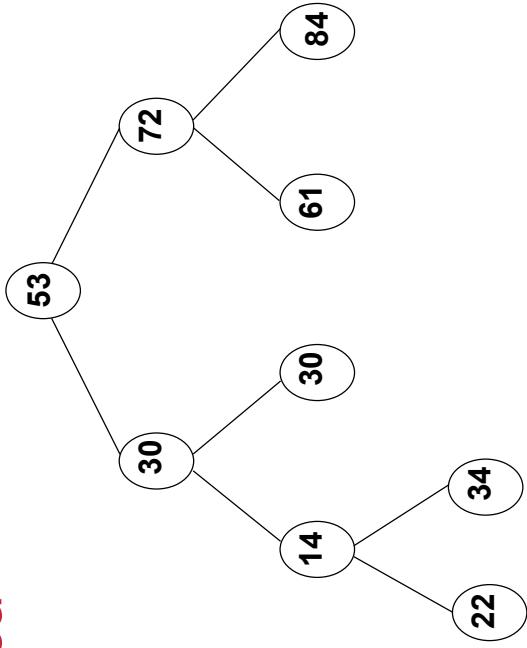


09/06/21

Datu-Egiturak eta Algoritmoak

34

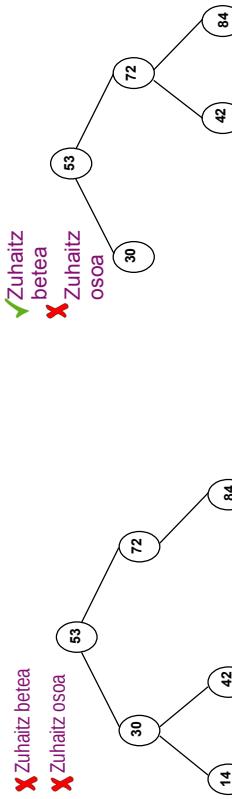
## Zuhaitz bitar osoa. Adibidea



09/06/21

Datu-Egiturak eta Algoritmoak

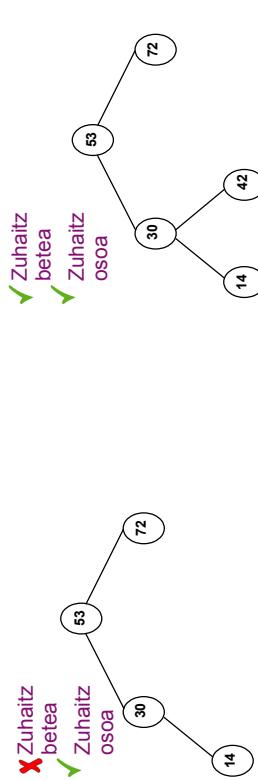
35



09/06/21

Datu-Egiturak eta Algoritmoak

35



09/06/21

Datu-Egiturak eta Algoritmoak

36

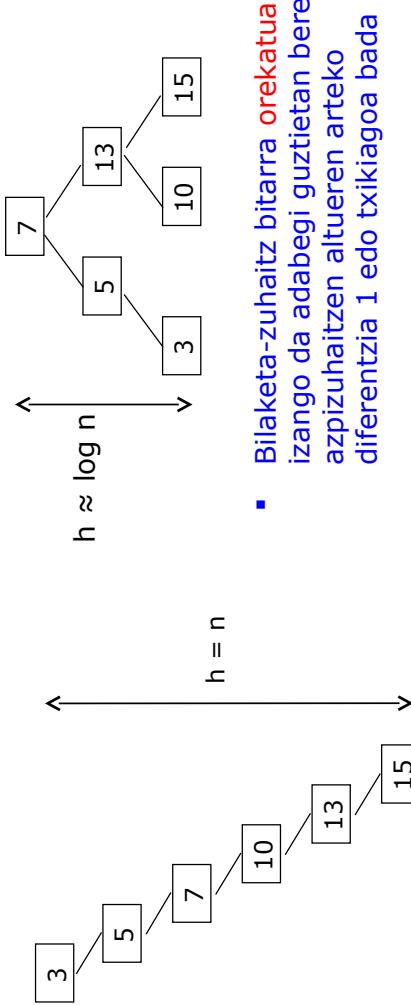
09/06/21

Datu-Egiturak eta Algoritmoak

36

# Bilaketa-zuhaitz bitar orekatuak

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



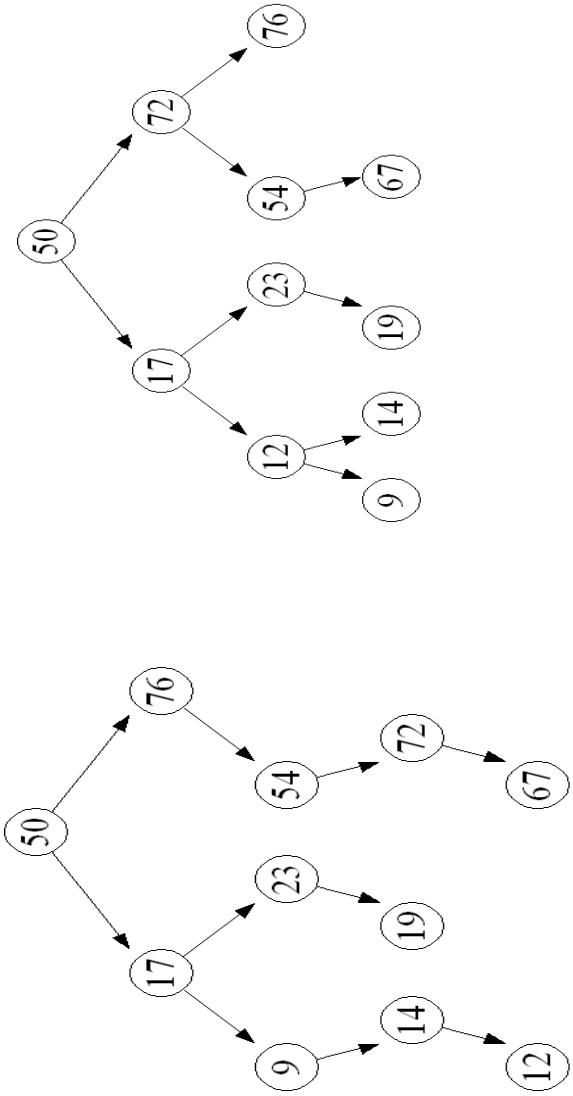
37

## Zuhaitz bitarren motak AVL zuhaitzak

- Zuhaitz hauen izena beren asmatzaileen deituretik dator: G.M. Adelson-Velsky eta E.M. Landis.
  - 1962an "An algorithm for the organization of information" izeneko artikuluan aurkeztu zuten datu-egitura hau.
- AVL zuhaitzak bilaketa-zuhaitz bitar orekatuak dira.**
  - AVL zuhaitzetan bilaketak, txertaketak eta ezabaketak  $O(\log n)$  dira. Hala ere, eransketek eta ezabaketek zuhaitza berrorekatzea eska dezakete. Zuhaitza berrorekatzeko errota zio bat edo gehiago egin behar izaten da.

## AVL zuhaitzak: adibideak

- Ez da AVL



09/06/21

Datu-Egiturak eta Algoritmoak

39

## e elementuaren txertaketa AVL batean

- Urratsak:

- 1.e elementuaren txertaketa, BZB arrunt batean bezala
- 2.e dagoen adabegiaren aurrekoen artean, ikusi zein den lehenengo adabegi desorekatua ( z ). y izango da z-ren umea e-rantz, eta x izango da z-ren biloba e-rantz
- 3.Orekatzu errrotatuz, z, y, eta x adabegiekin duten antoleraeraen arabera, 4 kasu bereizten dira:

- a)y z-ren ezkerreko umea da eta x y-ren eskueineko umea da (**Ezkerra Ezkuina Ezkerra Kasua**)
- b)y z-ren ezkerreko umea da eta x y-ren eskueineko umea da (**Ezkerra Ezkuina Kasua**)
- c)y z-ren eskueineko umea da eta x y-ren eskueineko umea da (**Eskuina Eskuina Kasua**)
- d)y z-ren eskueineko umea da eta x y-ren eskueineko umea da (**Eskuina Ezkerra Kasua**)

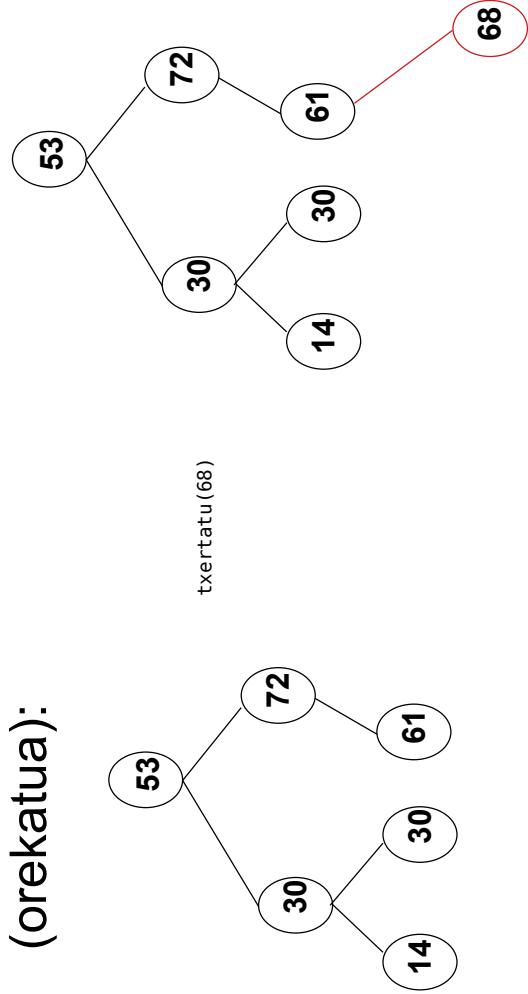
09/06/21

Datu-Egiturak eta Algoritmoak

40

## Txertaketa AVL-letan. Adibidea

- Jatorrizko zuhaitza  
(orekatua):



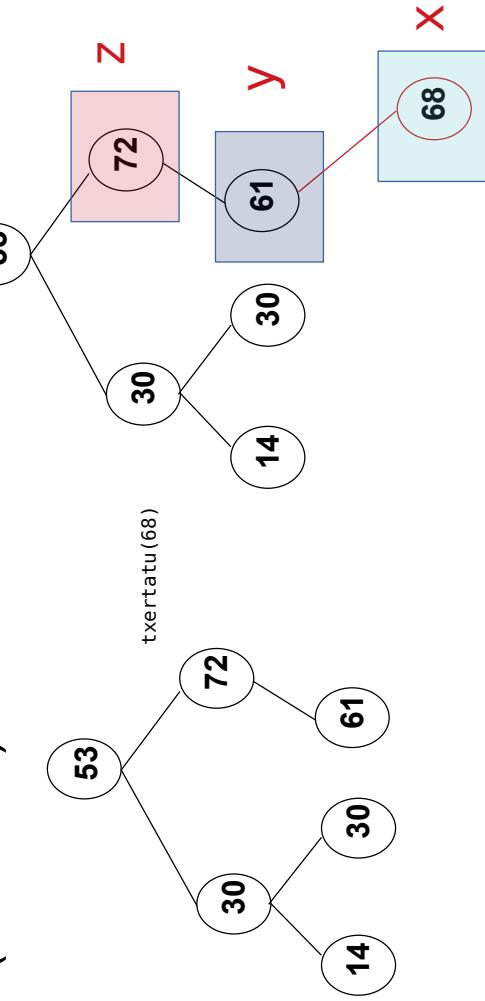
09/06/21

Datu-Egiturak eta Algoritmoak

41

## Txertaketa AVL-letan. Adibidea

- Jatorrizko zuhaitza  
(orekatua):

Ezkerra  
Eskuina kasua

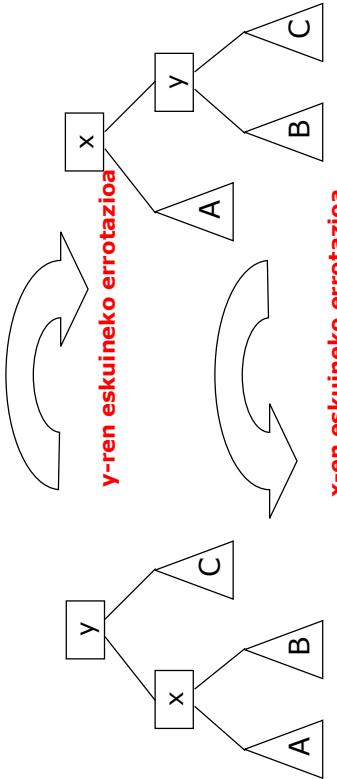
09/06/21

Datu-Egiturak eta Algoritmoak

42

## 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



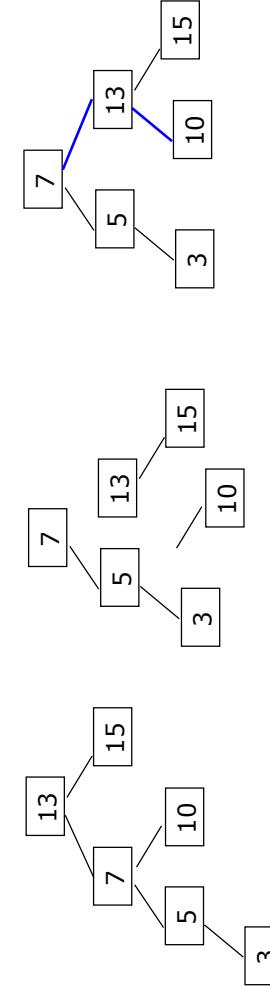
(errotazioa zuhaitzaren edozein adabegiri aplikatu ahal zaio)

Bilaketa-zuhaitz bitarrak

43

## Eskuineko errotazioaren adibidea

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



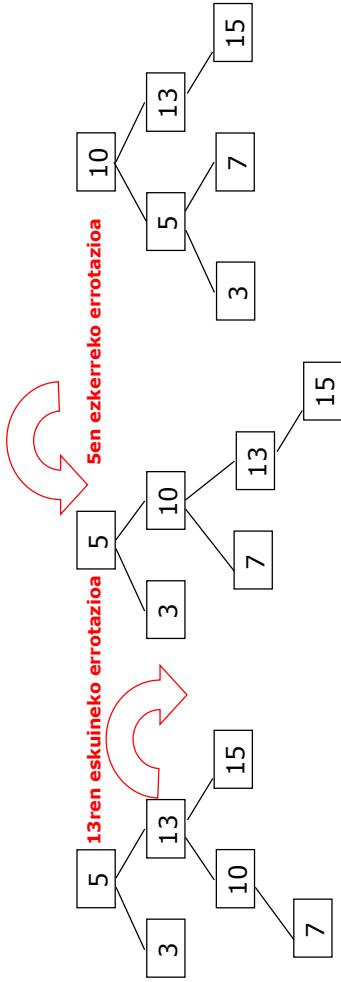
(Berdin ezkerreko errotazioan)

Bilaketa-zuhaitz bitarrak

44

## Eskuin-ezkerreko errotazioa

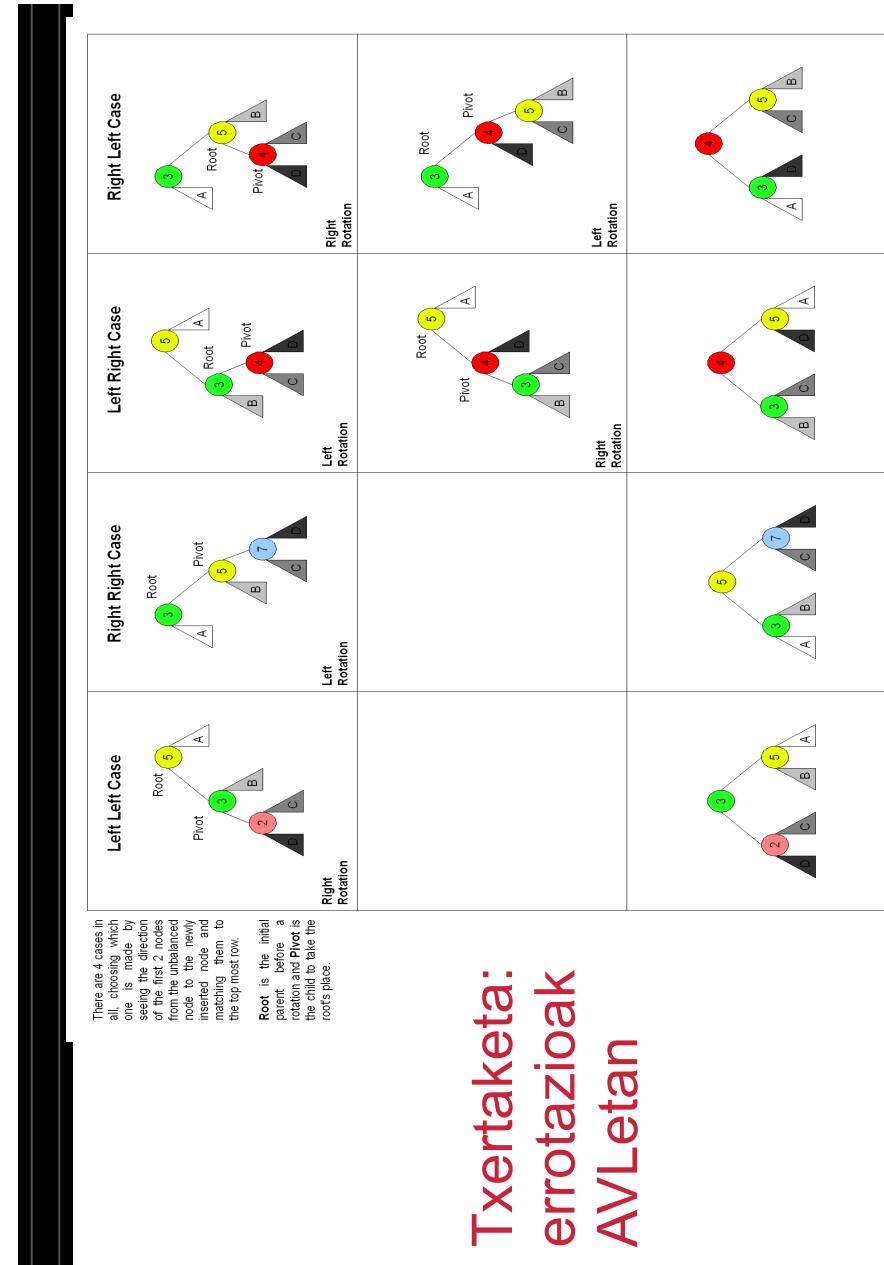
- Adar luzea eskuineko umearen ezkerreko azpizuhaitzean badago, orduan bi errotazio simple egin behar dira:
  - Lehenengo eskuineko errotazioa, erroaren eskuineko umeari
  - Ondoren erroaren ezkerreko errotazioa
- Mota honetako errotazioa egingo da 7 balioa duen adabegiaren txertaketaren ondoren



(Berdin ezker-eskuineko errotazioan)

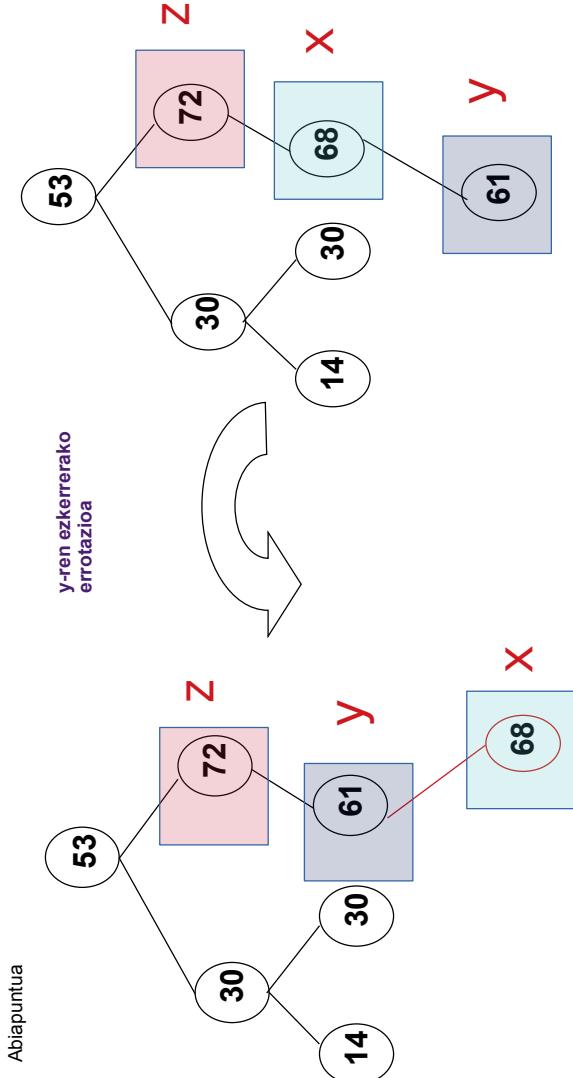
Bilaketa-zuhaltz bitarrak

45



**Txertaketa:  
errotazioak  
AVLetan**

## Txertaketa AVL-ean. Adibidea



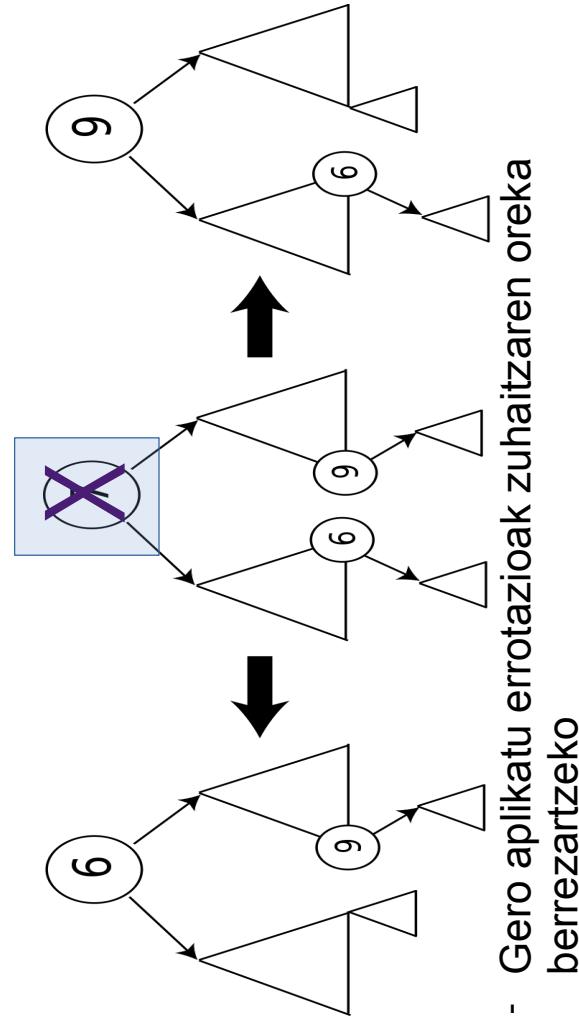
09/06/21

Datu-Egiturak eta Algoritmoak

47

## Ezabaketa AVL-ean

- Estrategia:
  - Ezabatu BZB arruntetan bezala



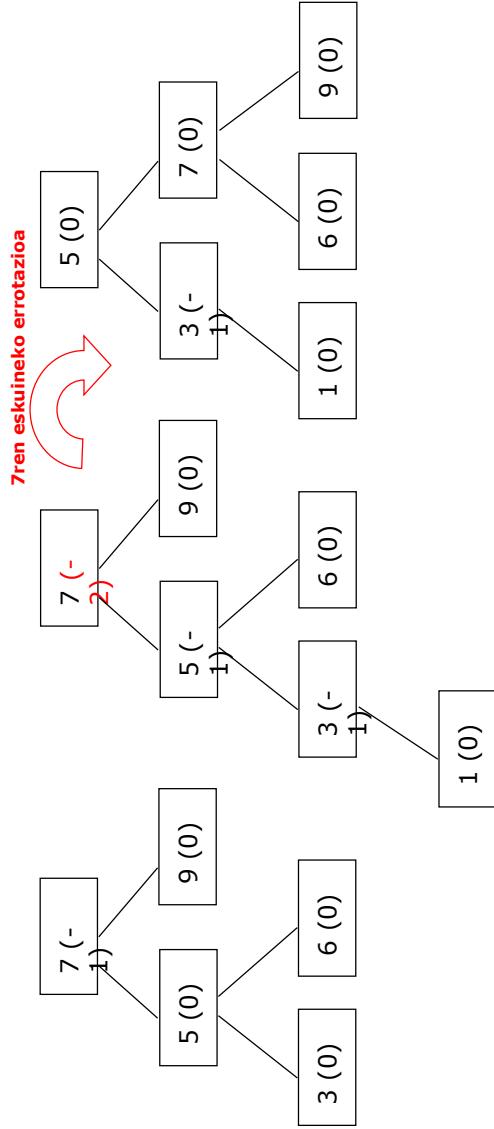
09/06/21

Datu-Egiturak eta Algoritmoak

48

# AVL zuhaitz baten orekatzea txertaketa baten ostean

Hasierako zuhaitza      Txertaketaren ondoren      Errrotazioaren ondoren



Bilaketa-zuhaitz bitarrak

49

## AVL zuhaitzak

- AVL zuhaitza Billaketa-zuhaitz bitarren implementazio orekatura da
- Adabegi batzen 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
- AVL zuhaitz batetako adabegiek oreka-faktorea eta gurasoaren esteka bat izan dezakete, orekatz-eragiketak azkartu ahal izateko

[Lewis eta Chase 2010]

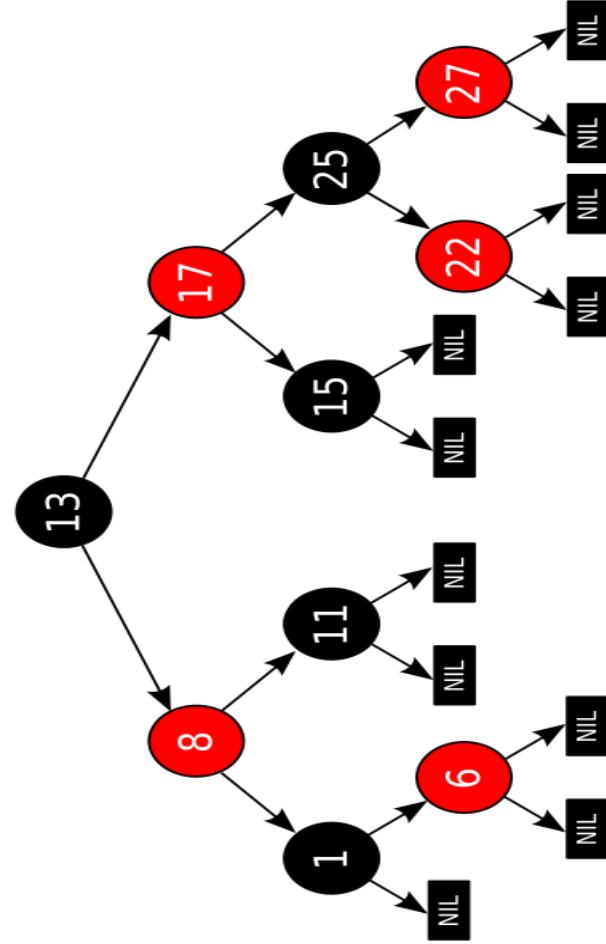
[http://en.wikipedia.org/wiki/AVL\\_tree](http://en.wikipedia.org/wiki/AVL_tree)

[http://en.wikipedia.org/wiki/Tree\\_rotation](http://en.wikipedia.org/wiki/Tree_rotation)

## Zuhaitz bitarren motak: zuhaitz gorri-beltzak

- Zuhaitz bitar bat **GorriBeltza (GB)** da, *red-black tree*, baldin eta soilik baldin honako baldintzak betetzen baditu:
  - adabegi guztiek *kolore* attributu bat dute, *gorri* edo *beltz* baloreak har ditzakeena,
  - bilaketa-zuhaitz bitarra da,
  - erroa beltza da,
  - hosto guztiak beltzak dira
  - gorria den edozein adabegik bi ume beltz ditu,
  - edozein adabegi hartuta, bere ondorengo hosto batera doan edozein bidek adabegi beltzen kopuru berdina du.

## Zuhaitz gorri-beltz baten adibidea



## Zuhaitz gorri-beltzen propietateak

- Errotik hostoetarako bide luzeena ez da bide motzena halako bi baino luzeagoa. Nolabait orekatuta dago
- Zuhaitz gorri-beltzetan bilaketak, txertaketak eta ezabaketak  $O(\log n)$  dira, kasu txarrean
- AVL zuhaitzten baldintzak zorratzagoak direnez, kasurik txarrean errotaazio gehiago egin behar dira AVL zuhaitzten zuhaitz gorri-beltzetan baino
- Hala ere, bilaketa intentsiboetarako AVL zuhaitzten portaera hobea da zuhaitz gorri-beltzena baino

09/06/21

Datu-Egiturak eta Algoritmoak

53

## Zuhaitz bitarren motak: meta egiturak, edo heap egiturak

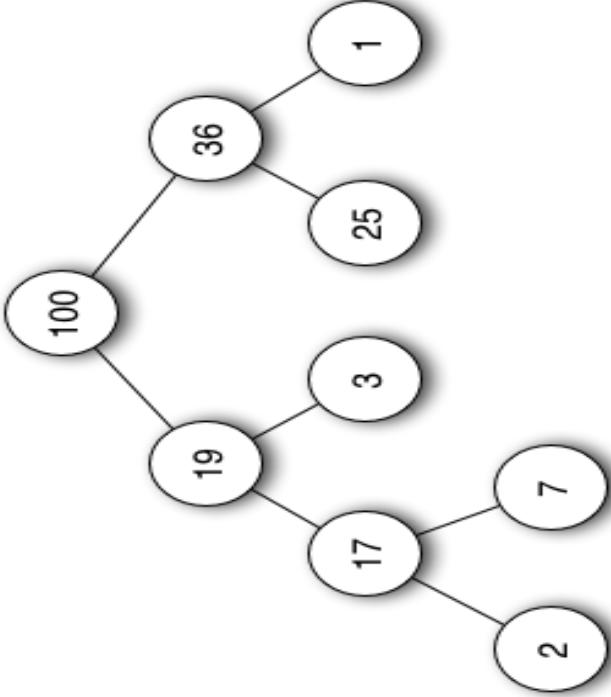
- Zuhaitz bitar osoak dira
- Propietateak:
  - umeen balioak gurasoen balioak baino txikiagoak edo berdinak dira beti (maximoen meta edo *Max-Heap*), edo
    - umeen balioak gurasoen balioak baino handiagoak edo berdinak dira beti (minimoen meta edo *Min-Heap*)
  - Egitura honek aplikazio ugari du. Batez ere, *heapsort* ordenazio-algoritmo famatua.

09/06/21

Datu-Egiturak eta Algoritmoak

54

## meta egituraren adibidea (max-heap)



09/06/21

Datu-Egiturak eta Algoritmoak

55

## B-tree egitura

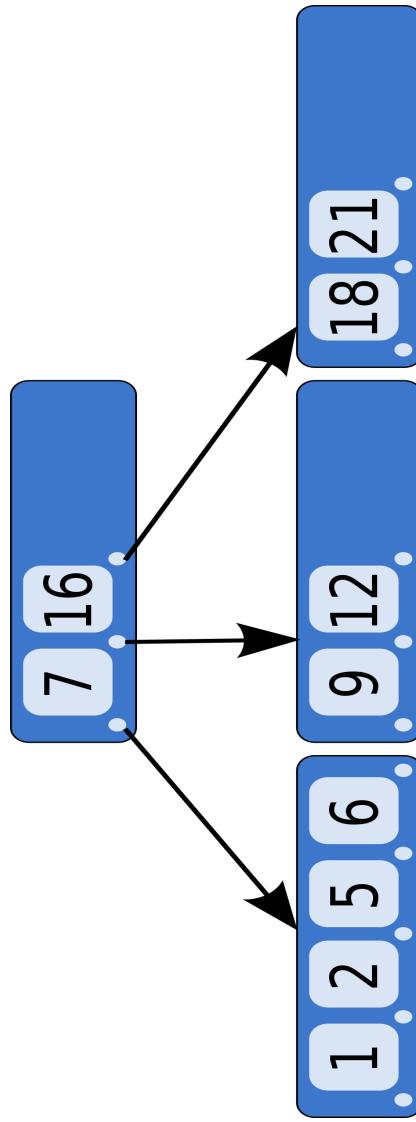
- m graduko *B-treea* honako propietateak betetzen dituen zuhaitza da:
  - Adabegi guztiak (erroak eta hostoek izan ezik)  $m/2$  ume dituzte gutxienez.
  - Erroak, hosto ere ez bada, 2 ume ditu gutxienez.
  - Hosto guztiak maila berean daude, eta ez dute gordetzen informaziorik.
  - K ume dituen barne-adabegi batek  $k-1$  gako ditu.
    - Gako horiek umeen gakoen banatzaila-rola jokatzen dute.

09/06/21

Datu-Egiturak eta Algoritmoak

56

## B-tree egitura baten adibidea



09/06/21

Datu-Egiturak eta Algoritmoak

57

## B-tree egituren propietateak

- Datu-kopuru handiak maneiatzeko aproposa da.
- Horregatik erabiltzen dira datu-baseetan eta fitxategi-sistemetan.
- Bilaketak:
  - Zuhaitz bitarren antzera
- Txertaketak:
  - Hostoetan egiten dira.
  - Txertaketak ume-kopuru maximoa gaindiarazten badu, hostoa bi nodutan banatu behar da.
- Ezabaketak:
  - Ezabaketa egindakoan, zuhaitza berregituratu egin behar da.

09/06/21

Datu-Egiturak eta Algoritmoak

58

## Trie

- Zuhaitz egitura ordenatua da
- n graduuko zuhaitzak direa (*n-tarrak*)
- Array *asoziatiboa* errepresentatzeko erabiltzen dira.
  - Gakoak *string*-ak izaten dira
  - Hitzen bildumetarako edo hiztegieta rako, bereziki

09/06/21

Datu-Egiturak eta Algoritmoak

59

## Trie-ak, hitzen bildumak errepresentatzeko

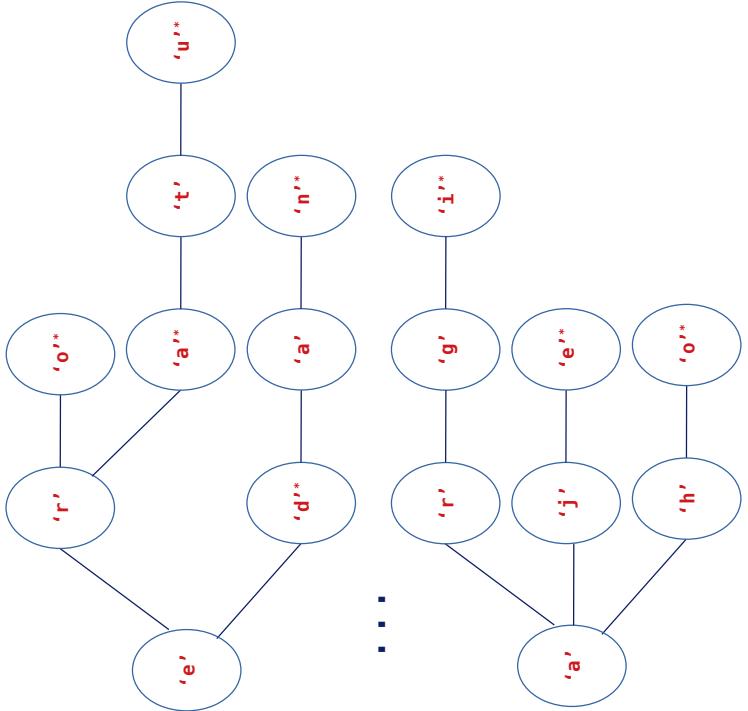
- Nodo batek alfabetoko karaktere adina ume eduki ditzake
- Errrotik hasi eta hitz-bukaerako marka duten nodoetarako bideek hitzak errepresentatzan dituzte

09/06/21

Datu-Egiturak eta Algoritmoak

60

## Trie egituraren adibidea



09/06/21

Datu-Egiturak eta Algoritmoak

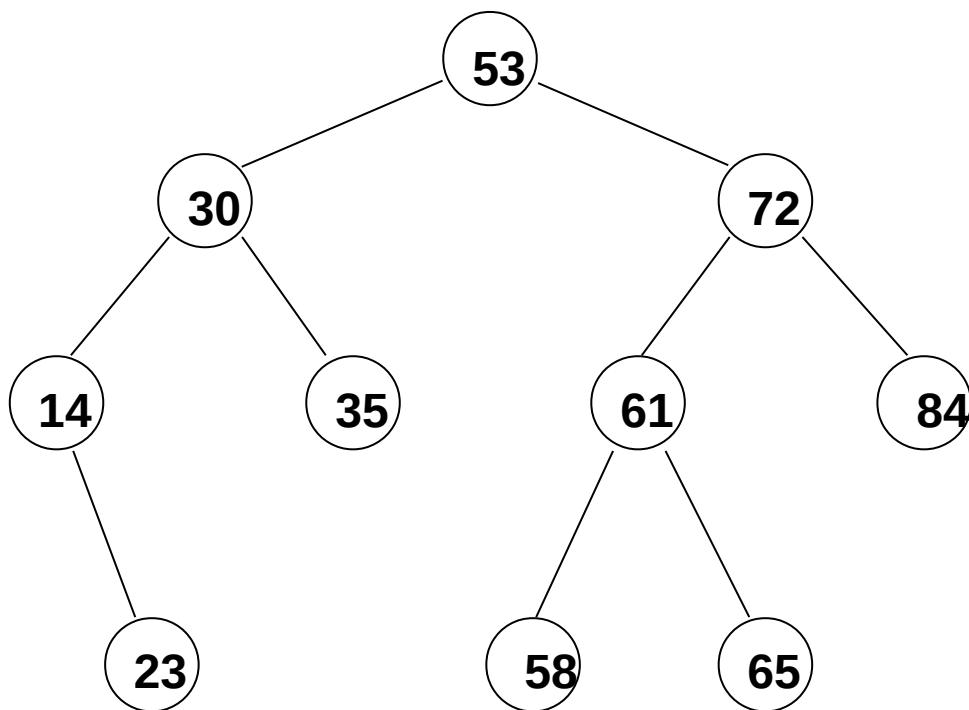
61

### ARIKETA 3.- BZB batean balioa gehitu eta tarteko balioak idatzia (1,5 puntu)

Bilaketa-zuhaitz bitar bat dugu, adabegiko balioaz gain, adabegi bakoitzak atributu gehigarri bat du azpizuhaitzak duen nodo kopurua adierazten duena (ikus `BinaryTreeNode` azpian).

Bi algoritmo hauek lortu nahi ditugu:

- `add(v)` metodoak v elementua zuhaitzean gehituko du, `numberOfNodes` eremua eguneratuz
- `printTartea(a, b)` metodoak a eta b tarteko balioak (a eta b barne) idatziko ditu, goranzko ordenan



```

public class BinaryTreeNode<T> {
 T data;
 BinaryTreeNode<T> left, right;
 int numberOfNodes;
}

public class BZB {
 BinaryTreeNode<Integer> root;

 public void add(int v)
 // Aurrebaldintza: v ez dago zuhaitzean.
 // Postbaldintza: v zuhaitzean txertatu da
 // Oharra: gogoratu txertaketaren ondorioz zenbait adabegiren numberOfNodes
 // atributua eguneratu beharko dela.

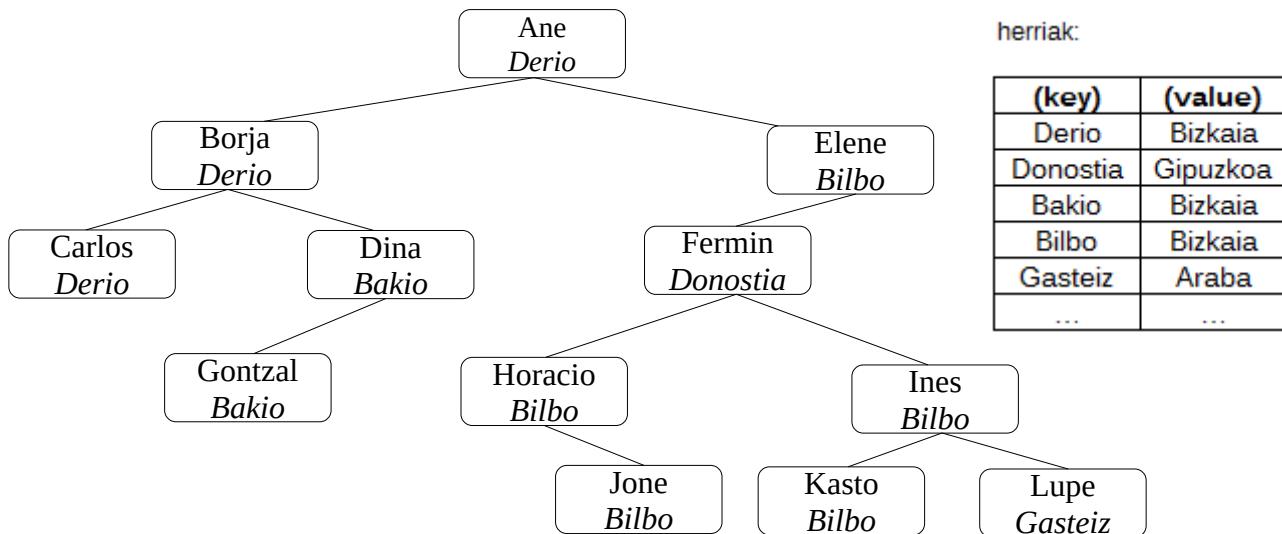
 public void printTartea(int a, int b)
 // Azalpena: Idatzi irteera estandarrean a eta b tarteko balioa duten nodoen balioak
 // (a eta b barne) goranzko ordenan.
 // Oharra: Errekurtsiboki implementatu eta soilik behar diren nodoak tratatu.
}

```

- `add(int v)` eta `printTartea(int a, int b)` metodoak implementatu
- Algoritmoaren kostuak kalkulatu, **modu arrazoituan**

### 3. ariketa: Zuhaitz genealogikoa (1,5 puntu)

Zuhaitz genealogikoa adierazteko zuhaitz bitar bat erabili dezakegu. Adabegi bakoitzak pertsona bat gordeko du, bere izena eta jaioterriarekin. Gainera adabegi bakoitzaren ezkerreko umeak pertsona horren aita nor den adieraziko du, eta eskuineko adabegiak pertsona horren ama. Adibidez, irudiko zuhaitzean, Ane Derion jaio zela eta bere gurasoak Borja eta Elene direla ikus dezakegu. Zenbait kasutan informazioa ez da osorik egongo (adibidez ez dakigu nor den Eleneren ama).



bizkaitarPetoPetoaDa metodoa implementatu behar duzu. Metodo honek bi parametro jasoko ditu: pertsona baten izena eta HashMap bat. Irudian ikus daitekeen moduan, HashMap-ean hainbat herriren izenak edukiko ditugu, bakoitza zein probintziatan dagoen adieraziz. Metodoak **pertsona hori bizkaitar peto-petoa den** kalkulatu behar du (eta **kostua adierazi** behar duzu). Pertsona bat bizkaitar peto-petoa izango da baldin eta honakoak betetzen badira:

- Zuhaitzean dago (suposatuko dugu zuhaitzean ez dagoela pertsona-izen errepikaturik)
- Bizkaiko herri batean jaio zen.
- Ezagunak diren bere arbaso guztiak ere Bizkaiko herri batean jaio ziren.

Adibidez, Borja bizkaitar peto-petoa da, bera eta bere arbaso ezagun guztiak (Carlos, Dina eta Gontzal) Bizkaiko herrian jaio zirelako. Ane berriz ez da bizkaitar peto-petoa, bere arbaso batzuk (Fermin eta Lupe) ez zirelako Bizkaian jaio.

```

public class Pertsona {
 String izena;
 String jaioterria;
}
public class BinaryTreeNode<T> {
 T data;
 BinaryTreeNode<T> left, right;
}
public class ZuhaitzGenealogikoa {
 BinaryTreeNode<Pertsona> root;

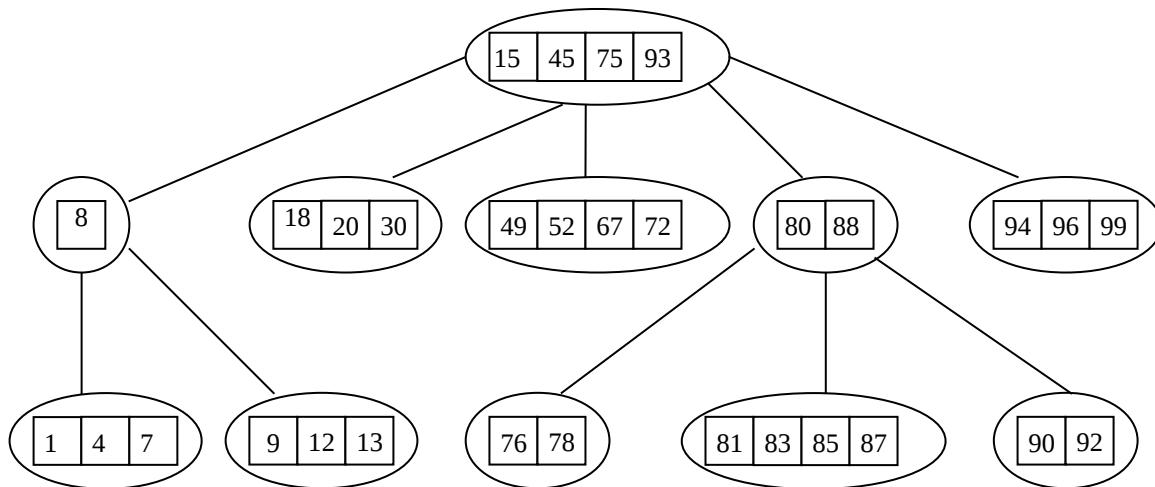
 public boolean bizkaitarPetoPetoaDa(String izena,HashMap<String, String> herriak){
 ...
 }
}

```

### 3. Bilaketa zuhaitz n-tarrean (1,5 puntu)

Bilaketa-zuhaitz N-tarra dugu, irudian ikusten den moduan. Hauek dira murrizpenak:

- Balio guziak desberdinak dira
- Adabegi bakoitzak N balio ditu gehienez, goranzko ordenan
- Adabegi batek N balio baldin baditu,  $a_1, a_2, \dots, a_n$ , orduan adabegi horrek  $N+1$  ume izango ditu. Ezkerreko umeak  $a_1$  baino txikiagoak diren balioak izango ditu; bigarren azpizuhaitzak  $a_1$  eta  $a_2$  bitarteko balioak izango ditu, ... eta azken azpizuhaitzak  $a_n$  baino handiagoak diren balioak izango ditu.



Hauek dira datu-erazagupenak:

```

public class BinaryTreeNode<T> {
 T[] balioak;
 BinaryTreeNode<T>[] umeak; // Balioen taulako tamaina + 1
}

public class Zuhaitza {
 BinaryTreeNode<Integer> root;

 public boolean badago(Integer elem)
 // post: emaitza true da "elem" zuhaitzean baldin badago eta
 // false bestela
}

```

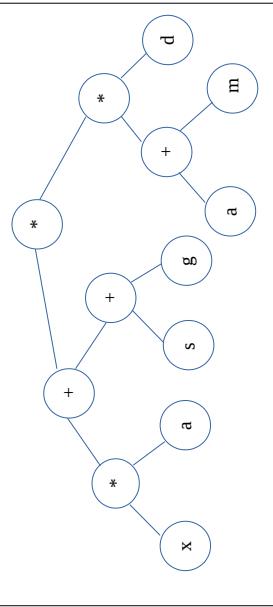
Hau eskatzen da:

- badago funtzioa implementatu
- Kalkulatu, modu arrazoituan, algoritmoaren kostua.

#### 4. Adierazpenaren ebaluazioa (2,5 puntu)

Azpirrograma bat egin nahi dugu, sarratzen zuhaitz bat eta hash-taula bat hartuko dituena. Bata espresio aritmetikoa adierazten duen zuhaitza da, eta bestea programa baten exekuzioaren une batean aldagaien balioak gordetzen dituena.

Espresio aritmetikoa bi adabegi-mota du: aldagaiak eta eragileak (simplifikatzearren, batuketa eta biderketa) eragileak bakarrik onartutako dira.



Zuhaitz horrek espresio hau adierazten du:  $((x * a) + (s + g)) * ((a + m) * d)$

Hash-taulak aldagaien izenak ditu gakotzat, eta bikote bakoitzak ondoko informazioa izango du:

|   |   |
|---|---|
| x | 4 |
| a | 5 |
| s | 7 |
| g | 1 |
| m | 5 |
| d | 2 |

Azpirrogramak espresio horen ebaluazioaren emaitza eman beharko du.  
Autoreko adibidean emaitza hau da: 560 =  $((4 * 5) + (7 + 1)) * ((5 + 5) * 2)$

Hau eskaaten da:

- Algoritmoa implementatu

- Algoritmaren kostua kalkulatu, modu arrazoitan.

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

public class InfoElemExpresioa {
 String elem; // *, +, edo aldagai baten izena
 boolean eragigai; // true -> eragigai, false -> aldagai
}

public class Zuhaitza {
 BinaryTreeNode<InfoElemExpresioa> root;

 public Integer evaluatu(HashMap<String, Integer> tHash)
 {
 // pre: tHash taulak aldagaiak ditu
 // post: Zuhaitzaren adierazpena evalutatu da.
 // Aldagaien balioak tHash taulatik hartu dira.
 // Aldagai bat ez badago hash-taulian, orduan
 // zero hartuko da bere baliortzat
 }
}
```

#### 4. Jokoaren zuhaitza (1,5 puntu)

Joko bateko informazioa gordetzeko zuhaitz bitar bat dugu. Jokalari bakoitzak puntuazio bat dauka.

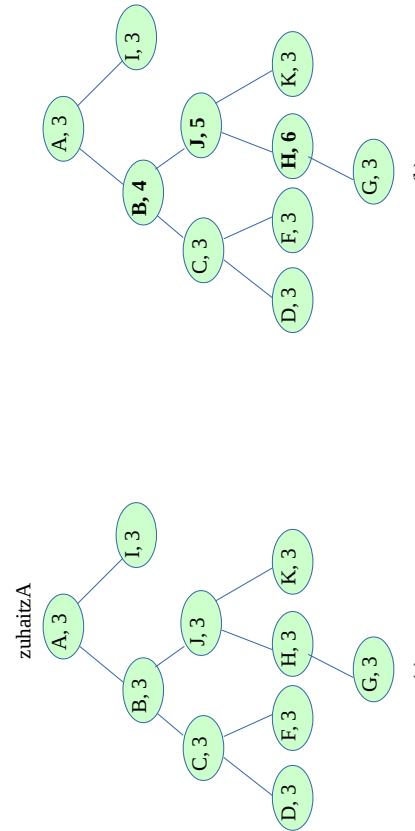
```
public class Info {
 String s;
 Integer puntuak;
}

public class Node {
 Info content;
 Node ezkerra, eskuina;
 Node gurasoa;
}

public class Zuhaitza {
 private Node root;
 public void saritu(int puntuak, String elem);
}
```

Jokalari batek puntuak lortzen dituenean, jokoak esaten du jokalari horri n puntu emango zaizkiola, eta puntuak banatutxo ditela bere arbosoaren artean: (n-1) puntu bere gurasori, (n-2) hurrengoan eta abar.

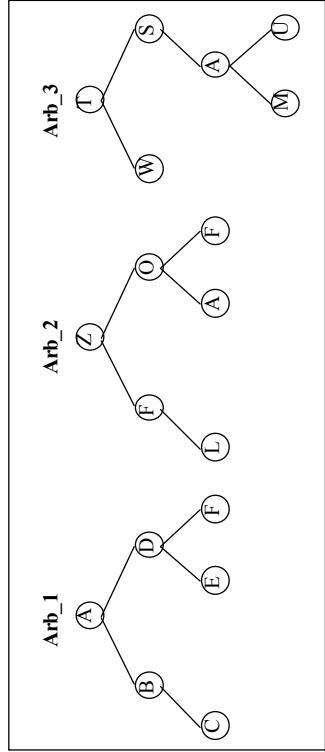
Adibidez, (a) irudiko zuhaitzean “Zuhaitza.saritu(3, “H”)” egingo bagenu, (b) irudiko zuhaitza emango luke, H-ni 3 puntu, J-ri 2 puntu eta B-ri puntu bat emanez.



Hau eskaaten da:

1. “saritu(…)” metodoaren implementazioa
2. Kalkulatu, modu arrazoituan, algoritmoaren kostua.

- 1. Egitura bereko zuhaitzak**  
 Bi zuhaitz bitar emenda, egituraBera izeneko funtzioa disenatu, zeinek true bueltatuko duen zuhaitzak egitura bera baldin badute (zuhaitzek egitura bera dute, adabegietako balioak ezik).
- Irudiko adibidean, egituraBera(Arb<sub>1</sub>, Arb<sub>3</sub>) deitak true bueltatuko du, eta egituraBera(Arb<sub>1</sub>, Arb<sub>3</sub>) deitak, aldiiz, false.



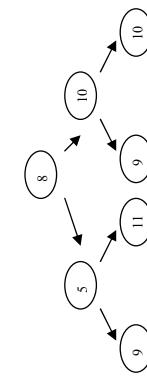
#### 2.- Maila bateko balio maximoa

Zuhaitz bitar bat eta zuhaitzko maila bat adierazten duen balio bate manda, apiprograma bat egin nahi dugu, maila horretako balio maximoa bueltatuko duena.

- Aziprograma egiteko, hau eskanzen da:
- Diseinu errekurtiboko pausoak
  - Implementazioa JavaZ

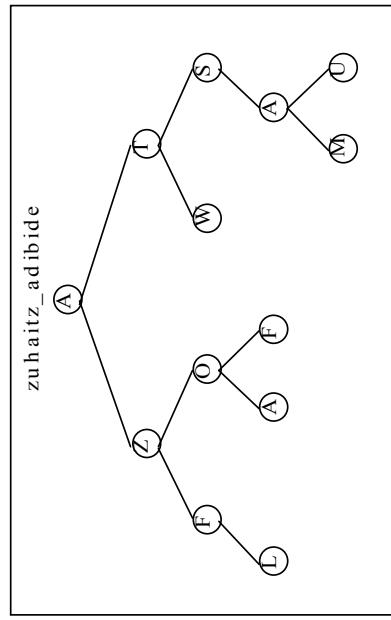
```
public int maxValor (BinTree<T> a, int n)
```

Adibidez, zuhaitz hau emanda:



#### 3. Adabegi-kopurua maila batean

- Disenatu eta implementatu JavaZ adabegiKop funtzioa, zeinek zuhaitz bitarra eta maila bat emanda, zuhaitz horretan maila horretan dagoen adabegi-kopurua bueltatuko duen.
- Irudian, adibidez:
- adabegiKop (zuhaitz\_adibide, 1) 1 bueltatuko luke
  - adabegiKop (zuhaitz\_adibide, 3) 4 bueltatuko luke
  - adabegiKop (zuhaitz\_adibide, 5) 2 bueltatuko luke



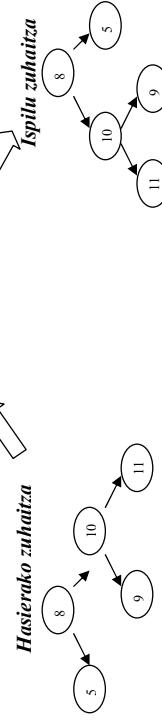
#### 4.- Zuhaitz ispliu

- Zuhaitz bitarra emanda, bere ispliu izango den bestie zuhaitz bat lortu nahi da.  
 Emaitzak adabegi berdinak izango ditu, isplitan ikusten diren bezala.

Ispliu funtziotan hasierako zuhaitza aldatu gabe egin nahi da. Soluzioan hau aurkeztuko da:

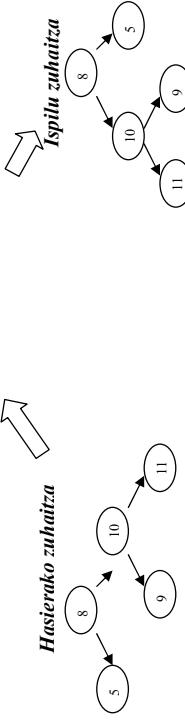
- Diseinu errekurtiboko pausoak
- Implementazioa JavaZ

```
public BinTree<T> ispliu (BinTree<T> a)
```



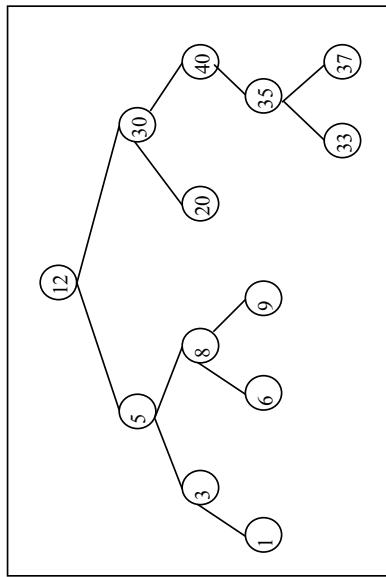
Adibidean ikusten da nola ekkerlean zeuden adabegiak eskuinera pasatu diren eta alderantziz.

$$\begin{aligned} \max\text{Valor } (a, 1) &= 8 \\ \max\text{Valor } (a, 2) &= 10 \\ \max\text{Valor } (a, 3) &= 11 \end{aligned}$$



### 5. Lista ordenatua

Azpiprograma bat diseinatu, bilaketa-zuhaitz bitar bat emanda, bere balioen zerrenda ordenatua bueltatuko duena.



```
public class LinkedList<T> listaOrdenatua ()
// pre:
// post: emaitza zuhaitzeko elementuen lista ordenatua da
```

Zerrendaren mota honela definitu da:

```
public class LinkedList<T> {
 LinkedException<T> first;
 LinkedException<T> last;

 public class LinkedException<T> {
 T data;
 LinkedException<T> next;
 }
}
```

Adibidez, irudiko zuhaitza emanda, lista hau lortuko da:

(1, 3, 5, 6, 8, 9, 12, 20, 30, 33, 35, 37, 40).

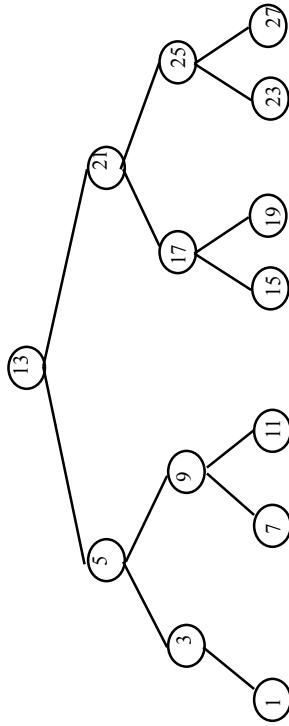
**Soluzioa denboralinealekoa izango da O(n).**

### 6.- Sortu BZB bat

Osokoen array bat emanda, bere balioak goranzko ordenan daudelarik, idatzi azpiprograma bat bilaketa-zuhaitz bitar orekatua lortzeko (hau da, ezkerreko eta eskuineko azpizuhaitzunten adabegi-kopurunen differentzia gehienez bat izango duena).

```
public BinTree<TP> surtuBZBorekatua (T[] taula)
// aurre: taula gorantz ordenatuta dago
// post: taula-k o "zuhaitza sortu da, BZBa da
// eta orekatuta dago azpizuhaitzetako adabegi-kopuruen arabera
```

Adibidez, taula = (1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27) emanda, emaitza ondokoak izango da:



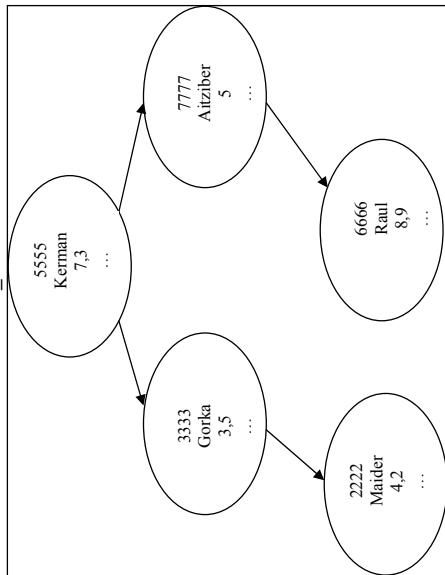
### 7. Listatu ordenatua hi Bilaketa Zuhaitz Bitarrekin

DEA irakasgaito ikasleei buruzko informazioa Bilaketa Zuhaitz Bitar bitan dago gordeta:

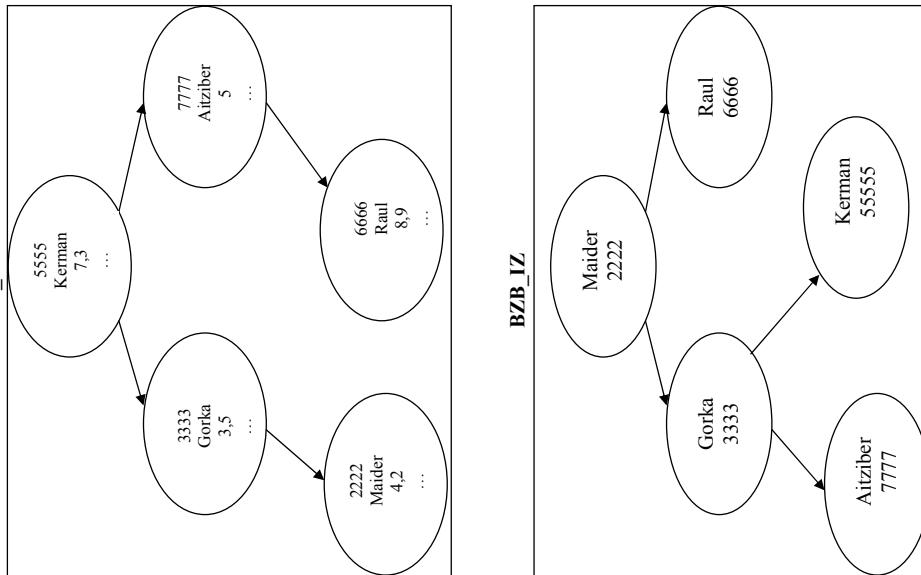
- BZB\_NA. Zuhaitz hau NA zenbakien arabera ordenatuta dago. Adabegi bakoitzean ikasle baten datuak (NA Zenbakia, Izena, Batezbesteko Nota, eta beste datu batzuk) dauzkagu.
- BZB\_IZ. Zuhaitz hau izenaren arabera ordenatuta dago. Zuhaitz honetako adabegi bakoitzean ikasle baten 2.datu (Izena eta NA Zenbakia) daukagu.

Adibidez:

**BZB\_NA**



**BZB\_IZ**



Hauek dira datu mota crazagupenak:

```

public class IkasleID {
 String izena;
 String na;
}

public class IkasleDatauk {
 IkasleID id;
 double nota;
}

```

#### Honako hau eskatzen da:

- a) Ondoko prozedura implementatu:

```

public void lortuListatuOrdenatua(BinSearchTree<IZ,
 BinSearchTree<NA>
 // aure: IZ bilaketa zuhaitz bitarra izenaren arabera ordenatuta dago.
 // NA bilaketa zuhaitz bitarra NA zenbakien arabera ordenatuta dago.
 // Zuhaitz biertan ikasle berberak daude
 // Ez dago ikasle-izen eurekipakurik.
 // post: Listatu ordenatua, izenaren arabera, pantailan idatzia.

```

Aurreko adibidea kontuan hartuta, hau izango litzateke lortu behar den listatu:

| Izena    | NA Zenbakia | Batezbesteko Nota |
|----------|-------------|-------------------|
| ...      | ...         | ...               |
| Aitziber | 7777        | 5                 |
| Gorka    | 3333        | 3,5               |
| Kerman   | 5555        | 7,3               |
| Maiter   | 2222        | 4,2               |
| Raul     | 6666        | 8,9               |

- b) Kalkulatu, modu arrazoituan, algoritmoaren kostua. Aldagaiaik erabiliz gero, aldagai bakotza zer den adierazi behar da.

### 8. Deskargaren bilaketa zuhaitz bitarra (2,5 puntu)

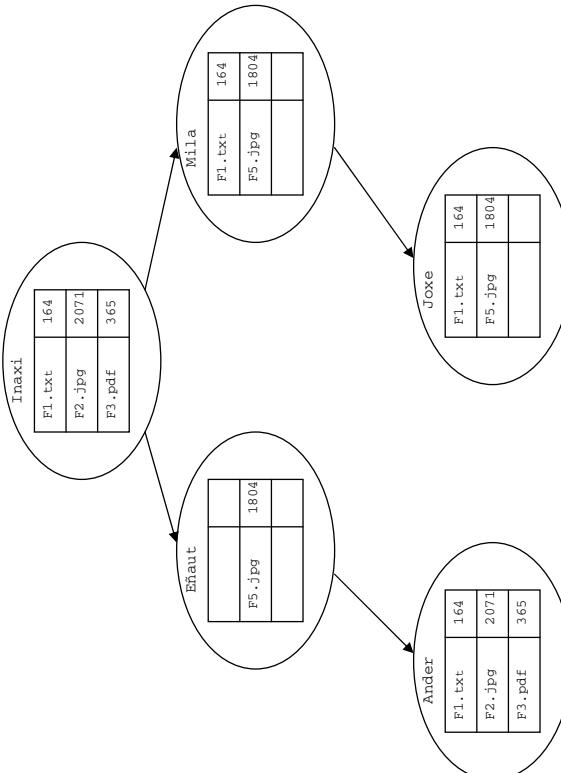
Erbilizaleek deskargatutako fixategiek kudeatzeko *Bilaketa-Zuhaitz Bitar* bat eman zaigu. Adabegi bakoitzean honako informazio hau dago:

- Erabilizalearen izena.
- Hash Taula bat, non deskargatutako fixategien informazioa dagoen: fixategien gakoa (gako gisa erabilizilen dena), fixategaren izena eta tamaina (zenbat KB-takoak den).

Bilaketa-Zuhaitz Bitarra erabilizale-izenaren arabera dago ordenatua. Aurre-baldintza da ez degoela erabilizale-izenen etrepikaturik. Honako eskatzan da:

- Definitu Adan datu-egitura hori.
  - Especifikatu, diseinatu eta implementatu Adan azpiprograma bat, fixategi-izen bat emanda, bi emaitza hauek itzultzen dituena:
- Zenbat aldiz deskargatu den fixategi hori, eta
  - Zerrenda estekatu bat fixategi hori deskargatu duten erabilitzaleen izenekin.

Adibidez:



Adibide honetan, "F5.jpg" fixategi-zena emanda, honako emaitzak iztuli beharko lituzke azpiprogramak:

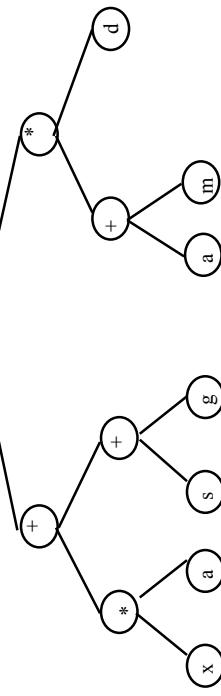
- Deskarga-kopurua: 3
- Erabilizaleen zerrrenda: >"Enaut", "Joxe", "Mila">

### 9.- *Expresio aritmetikoa ebaluatu*

Azpiprograma bat egin nahi dugu, sartzerat 2 zuhaitz hartuko dituena. Bata expresio aritmetikoa adierazten duen zuhaitza da, eta bestea programa baten exekuzioaren une batean aldagaien balioak gordetzen dituena.

- Expresio aritmetikoa bi adabegi-mota ditu: aldagaiak eta eragileak (simplifikatzearren, batuketa eta biderketarako eragileak bakarrak onartuko dira).

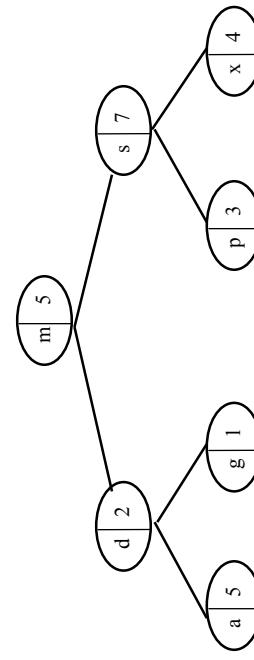
```
public class InfoElemExpresioa {
 String elem; // *, +, /, -, <, >, ==, !=, <=, >=
 boolean operador; // true -> eragilea, false -> aldagai
}
```



Zuhaitz horrek expresioa hau adierazten du:  $((x * a) + (s + g)) * ((a + m) * d)$

Aldagaien balioak daturkan zuhaitza bilaketa zuhaitz bitarra da, aldagaiaren arabera ordenatuta, eta adabegi bakoitzak ondoko informazioa du:

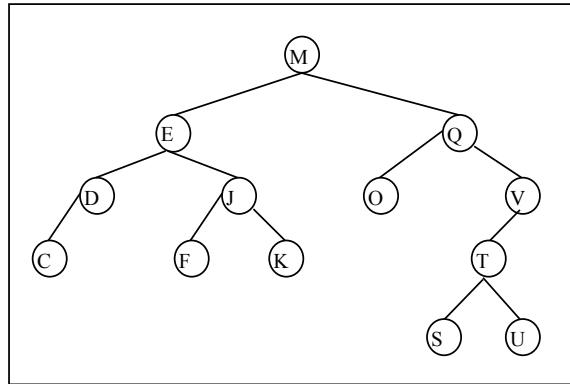
```
public class InfoAldagaiia {
 String nom;
 int valor;
}
```



- Azpiprogramak expresio horren ebaluazioaren emaitza eman beharko du..
- Aurreko adibidean emaitza hau da: 560 = ((4 \* 5) + (7 + 1)) \* ((5 + 5) \* 2)

### 3. Eraginkortasunaren kalkulua (2,5 puntu)

Bilaketa-zuhaitz bitarra dugu, adabegi bakoitzak String motako elementua duelarik. Bilaketan aztertuko den batezbesteko elementu-kopurua kalkulatu nahi da, eta horretarako zenbatu egindo dira zerrenda bateko elementuen bilaketan aztertuko diren elementuak.



Funtzio hau implementatu nahi da:

```

public class BinaryTreeNode<T> {
 protected T content;
 protected BinaryTreeNode<T> left;
 protected BinaryTreeNode<T> right;
}

public class BinarySearchTree<T> {
 protected BinaryTreeNode<T> root;
 protected int count;
}

public class NireZuhaitza extends BinarySearchTree<String> { // Herentzia

 public float batezbestekoElementuak(SimpleLinkedList<String> l) {
 // Aurre: l listako elementu guztiek zuhaitzean daude
 // Post: emaitza l zerrendako elementuak bilatzean aztertutako
 // elementuen batezbestekoa izango da
 }
}

```

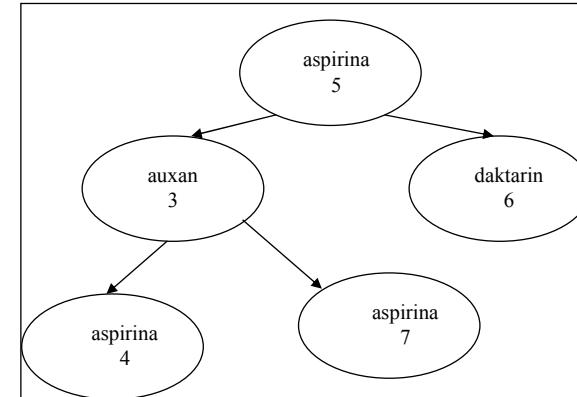
Adibidez, sarrerako zerrenda hau balitz: <T, M, E, S>, emaitza hau izango litzateke:

- 4 elementu aztertuko dira T bilatzeko
- 1 elementu aztertuko da M bilatzeko
- 2 elementu aztertuko dira E bilatzeko
- 5 elementu aztertuko dira S bilatzeko

Eta 3 izango da azken emaitza =  $(4 + 1 + 2 + 5) / 4$

### 4. Zuhaitzaren bihurketa (1,5 puntu)

Produktuen salmentak dituen zuhaitz bitarra emanda (adabegi bakoitzean (produktua, salduztako unitateak) moduko bikotea), zuhaitz horretako elementuak hash taula batera pasako dituen azpiprograma nahi dugu, produktu bakoitzeko elementu bakarra emanet, produktu horren salmenta guztiekin.



Adibideko zuhaitza emanda, lortuko den hash taulak hau izango luke:

|   |          |
|---|----------|
| 0 |          |
| 1 | daktarin |
| 2 |          |
| 3 | aspirina |
| 4 | auxan    |
| 5 |          |

```

public class BinaryTreeNode<T> {
 protected T content;
 protected BinaryTreeNode<T> left;
 protected BinaryTreeNode<T> right;
}

public class BinaryTree<T> {
 protected int count;
 protected BinaryTreeNode<T> root;
}

public class Produktu {
 int salmentak;
 String izena;
}

public class ProduktuenZuhaitza extends BinaryTree<Produktu> {

 public HashMap<String, Integer> zuhaitzaHTBihurtu()
}

```

#### 4. N hoberenak lortu (2,5 puntu)

M balio positiboen zerrenda dugu ( $M = 1.000.000.000$ ) eta zerrendako N balio handienak lortu nahi dira ( $N < M$ , adibidez,  $N = 1.000$ ).

```
public ArrayList<Integer> nHoberenakLortu(ArrayList<Integer> lista)
```

4 ikasleri galdeztuta, soluzio hauek eman dizkigute:

##### 1. soluzioa)

```
emaitza = zerrenda hutsa
errepikatu N aldiz
bilatu "lista"ko maximoa
sartu maximoa emaitza zerrendan
aldatu "lista"ko balio maximoa -1 balioagatik (horrela,
hurrengoan ez da izango maximoa)
```

##### 2. soluzioa)

```
"lista" beheranzko ordenan ordenatu
hartu lehen N balioak eta bueltatu
```

##### 3. soluzioa)

```
emaitza = zerrenda hutsa
bilaketaZuhaitzBitarra -> sartu "lista"ko M balioak
errepikatu N aldiz
zuhaitzeko balio handiena bilatu eta ezabatu
sartu balio handiena emaitza zerrendan
```

##### 4. soluzioa)

```
bilaketaZuhaitzBitarra = zuhaitz hutsa
jatorrizko zerrendako elementu bakoitzeko (M aldiz)
if zuhaitzaren tamaina < N
then sartu elementua zuhaitzean
else if zuhaitzaren tamaina = N and
 zuhaitzeko balio txikiena < uneko elementua
 then ezabatu zuhaitzeko elementu txikiiena
 Gehitu zuhaitzean uneko elementua
bueltatu bilaketaZuhaitzBitarra-ko elementuak
```

Hau eskatzen da:

1. Lau algoritmoak ordenatu beraien konplexutasunaren arabera, **algoritmo bakoitzeko bere kostua kalkulatz, era arrazoiutan.**
2. Implementatu Javaz algoritmo eraginkorrena.

# Grafoak

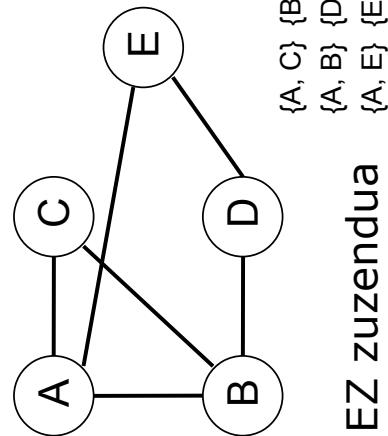
Lengoaia eta sistema informatikoak  
UPV-EHU

Jesús Bermudez-en gardenkietan oinarrituta  
Eta haren Creative Commons baimenekin  
zabaldua

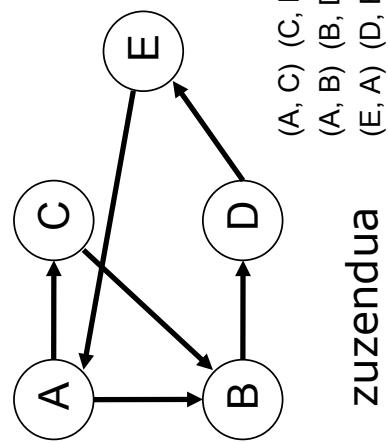


## Zer da grafo bat?

- $G = (N, A)$ 
  - N adabegien multzo bat da (edo erpinak)
  - A arkuen multzoa (adabegi bikotez adieraziak)



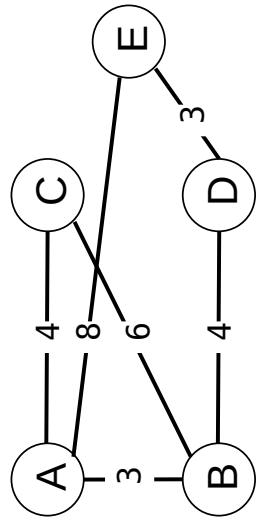
|             |            |            |            |            |            |            |
|-------------|------------|------------|------------|------------|------------|------------|
| EZ zuzendua | $\{A, B\}$ | $\{A, E\}$ | $\{B, C\}$ | $\{B, D\}$ | $\{C, E\}$ | $\{D, E\}$ |
|             |            |            |            |            |            |            |



|        |        |
|--------|--------|
| (A, C) | (C, B) |
| (A, B) | (B, D) |
| (E, A) | (D, E) |

## Oinarrizko grafoen motak

- **Ponderatua:** arkuek nolabaiteko *pisua* edo *balioa* dute.
  - Pisu horrek esanahi gehigarri bat ematen dio arkuari.



08/09/2022

Datu-Egiturak eta Algoritmoak

3

DEA

## Zenbait kontzeptu

- Auzokidetasuna: bi adabegien artean auzokidetasun erlazioa dagoela esango dugu, baldin eta bi adabegi horiek lotzen dituen arkurik badago.
- Ibilbidea: grafoko bi adabegi lotzen dituen arkuen sekuentzia bat da.
- Zikloa: hasten den adabegi berean bukatzen den ibilbidea da, non ez baita arkurik errepikatzen.

## Zenbait kontzeptu

- Grafoak vs zuhaitzak: grafoa zuhaitza baino kontzeptu orokorragoa da, ez baita kontuan hartzen zuhaitzen murritzapen hau:
  - Adabegi bakoitzak guraso bakarra du (erroak izan ezik, zeinak ez baitu aitarik)
- Grafoetan ez dago errorik, eta adabegi bakoitza konektatua egon daiteke gainontzeko n-1 adabegietara (gehienez).
- Grafo bat osoa dela esaten da adabegiak konektatzen dituzten arkuen kopurua maximoa baldin bada.

---

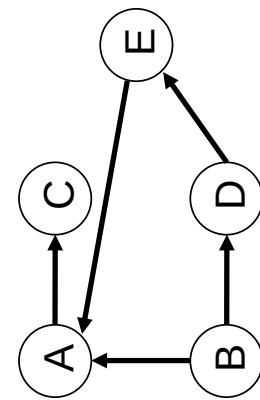
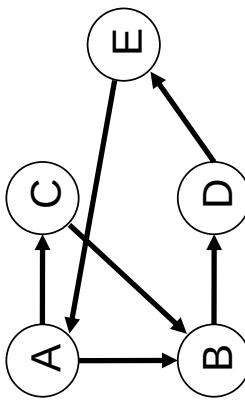
Grafoak/Grafos

5

## Konektitatea

- Bi erpin **konektatuta** daude batetik besterako ibilbide bat baldin badago.
- **Grafo** bat **konekxua** (**konektatua**) da baldin eta bi edozein erpin hartuta, bi erpin horiek lotzen dituen ibilbide bat badago.
- Grafo zuzenduetan:
  - Grafo bat **ahul konektatua** (*weakly connected*) da baldin eta bere arku zuzendu guztiengordez arku ez-zuzenduak jarrita grafo konexua (ez-zuzendua) lortzen bada.
  - Grafo bat **hertsiki konektatua** (*strongly connected*) da baldin eta edozein u, v erpinak hartuta, u-tik v-ra eta v-tik u-ra doazen bide zuzenduak baditu.
  - Grafo bat **erdikonektatua** (*semiconnected*) da baldin eta edozein u, v erpinak hartuta, u-tik v-ra edo v-tik u-ra doan bide zuzenduren bat badu.

## Konektibitatea. Adibideak

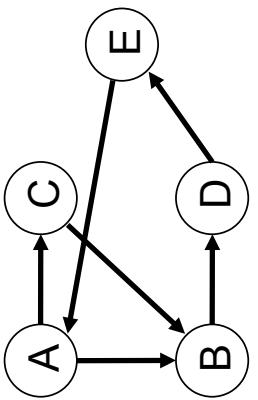
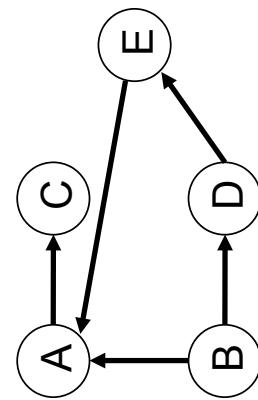


08/09/2022

Datu-Egiturak eta Algoritmoak

7

## Konektibitatea. Adibideak

**Hertsiki konektatua****Erdi konektatua**

08/09/2022

Datu-Egiturak eta Algoritmoak

8

**Ahul konektatua**

7

# GraphADT (interface)

|                                        |                                                                 |
|----------------------------------------|-----------------------------------------------------------------|
| void addVertex (T vertex)              | Adds a vertex to this graph                                     |
| void removeVertex (T vertex)           | Removes a single vertex with the given value from this graph    |
| void addEdge (T vertex1, T vertex2)    | Inserts an edge between two vertices of this graph              |
| void removeEdge (T vertex1, T vertex2) | Removes an edge between two vertices of this graph              |
| Iterator iteratorBFS(T startVertex)    | Returns a breadth first iterator starting with the given vertex |
| Iterator iteratorDFS(T startVertex)    | Returns a depth first iterator starting with the given vertex   |
| boolean isConnected()                  | Returns true if this graph is connected, false otherwise        |

Grafoak/Grafos

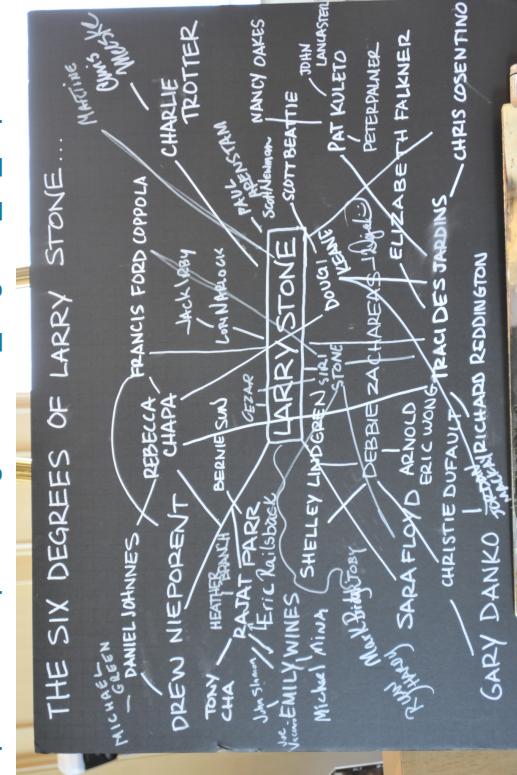
9

## Aplikazioen adibideak.

### Pertsonen arteko distantziaren maila

- Six degrees of separation:

[https://en.wikipedia.org/wiki/Six\\_degrees\\_of\\_separation](https://en.wikipedia.org/wiki/Six_degrees_of_separation)



08/09/2022

Datu-Egiturak eta Algoritmoak

10

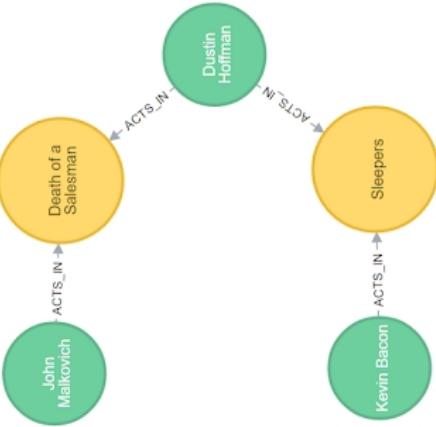
# Aplikazioen adibideak.

## Kevin Baconen zenbakia

- Six Degrees of Kevin Bacon edo Kevin Baconen zenbakia:

[https://en.wikipedia.org/wiki/Six\\_Degrees\\_of\\_Kevin\\_Bacon\\_numbers](https://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon_numbers)

- Zein da John Malkovichen Kevin Bacon zenbakia? (distantzia minimoa)



08/09/2022

Datu-Egiturak eta Algoritmoak

11

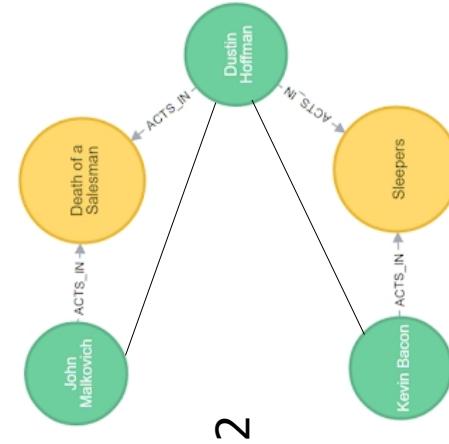
# Aplikazioen adibideak.

## Kevin Baconen zenbakia

- Six Degrees of Kevin Bacon edo Kevin Baconen zenbakia:

[https://en.wikipedia.org/wiki/Six\\_Degrees\\_of\\_Kevin\\_Bacon\\_numbers](https://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon_numbers)

- Zein da John Malkovichen Kevin Bacon zenbakia? (distantzia minimoa)



2

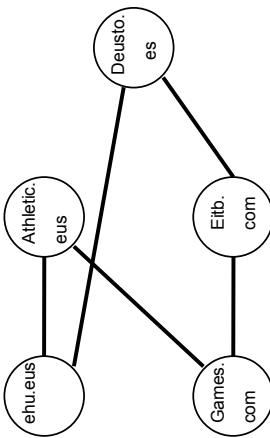
08/09/2022

Datu-Egiturak eta Algoritmoak

12

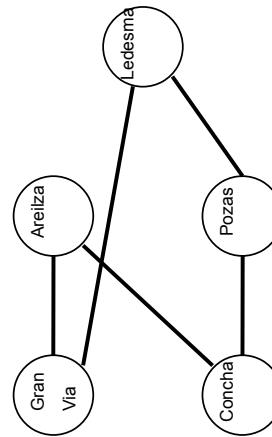
## Aplikazioak

- Internet: bi web-orri konektatzen dituzten estekak bilatu:



## Aplikazioak

- Bideak bilatu:
  - Nola joan Gran Vía-tik Autonomía-ra?  
Gran Vía → Máximo Agirre → Areilza → Autonomía



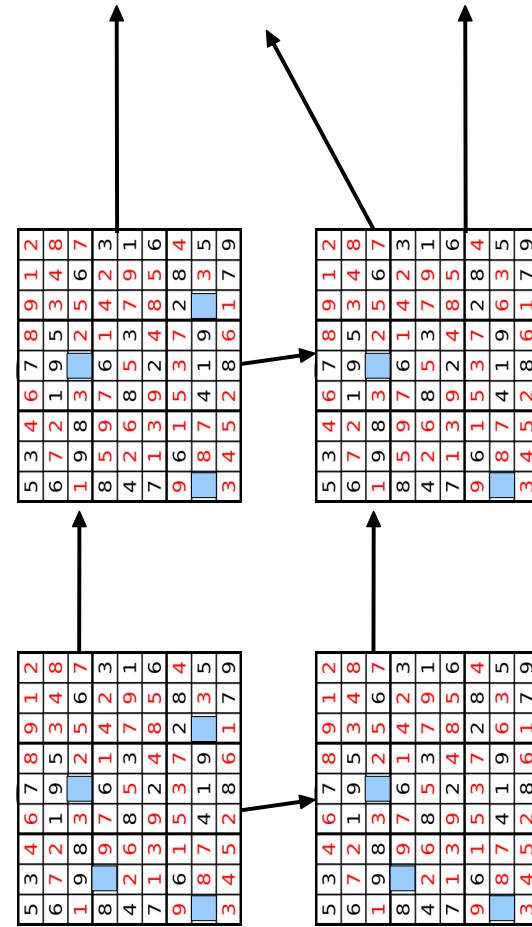
Metro sistema



Aplikazioak

DEA

- Jokoak eta adimen artifiziala: sudoku

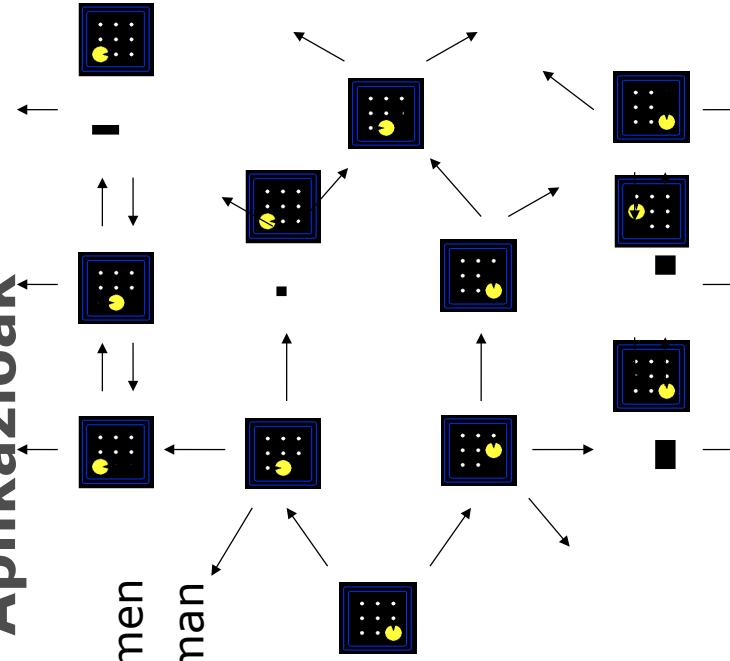


Grafoak/Grafos

16

## Aplikazioak

- Jokoak eta adimen artifiziala: pacman

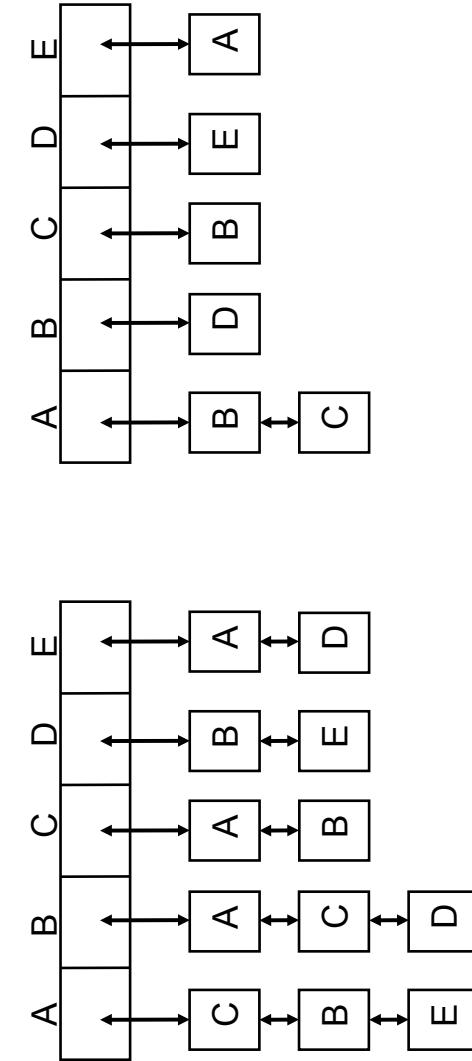


Grafoak/Grafos

17

## Grafoen adierazpideak

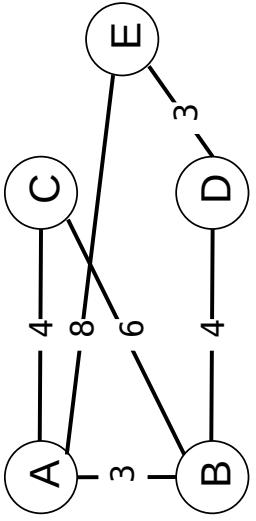
- Auzokidetasun-zerrendak



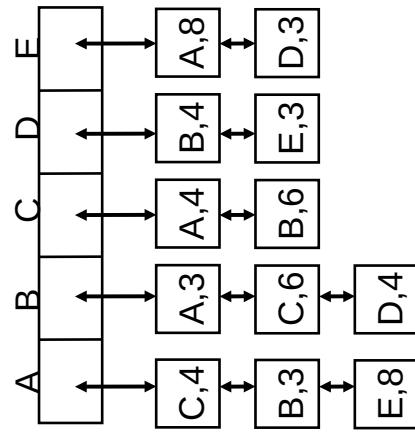
Grafoak/Grafos

18

# Auzokidetasun-Zerrendak



Grafo ez-zuzendu ponderatua



08/09/2022

Datu-Egiturak eta Algoritmoak

19

DEA

## Grafoen adierazpideak

### ■ Auzokidetasun-Zerrendak

```
public class GraphAL<T> implements GraphADT<T>
{
 protected final int DEFAULT_CAPACITY = 10;
 protected int numVertices; // number of vertices in the graph
 protected LinkedList<Integer>[] adjList; // adjacency list
 protected T[] vertices; // values of vertices
}

public GraphAL() { // Eraikitzalea }
```

# Grafoen errepresentazioa Map erabiliz

- Auzokidetasun-matrizetan nahiz - listetan, erpinen etiketak (gakoak) eta beren biltegiratzे-posizioak (indizeak) erpinak array baten bidez gestionatzendira.
- Beste aukera bat da **Map** hurbilpena erabiltzea.

08/09/2022

Datu-Egiturak eta Algoritmoak

21

## Grafoen errepresentazioa Map + auzokidetasun-lista erabilita

```
public class GrafoPonderatuMap() { // Eraikitzailea
 protected HashMap<Erpina,
 LinkedList<ErpinPonderatua>> auzoLista;

 public GrafoPonderatuMap() { // Eraikitzailea
 this.auzoLista = new HashMap<Erpina,
 LinkedList<ErpinPonderatua>>();
 }
}
```

08/09/2022

Datu-Egiturak eta Algoritmoak

22

## Grafoen adierazpiak

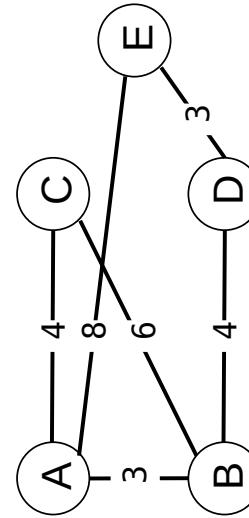
### Auzokidetasun-matrizea

|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 | 1 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 0 | 0 |
| D | 0 | 1 | 0 | 0 | 1 |
| E | 1 | 0 | 0 | 1 | 0 |

Grafoak/Grafos

23

## Auzokidetasun-matrizea



Grafo ez-zuzendu ponderatua

|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 3 | 4 | 0 | 8 |
| B | 3 | 0 | 6 | 4 | 0 |
| C | 4 | 6 | 0 | 0 | 0 |
| D | 0 | 4 | 0 | 0 | 3 |
| E | 8 | 0 | 0 | 3 | 0 |

## Grafoen adierazpiak

- Auzokidetasun-matrizea

```
public class Graph<T> implements GraphADT<T>
{
 protected final int DEFAULT_CAPACITY = 10;
 protected int numVertices; // number of vertices in the graph
 protected boolean[][] adjMatrix; // adjacency matrix
 protected T[] vertices; // values of vertices

 public Graph() { // Eraikitzailea
 ...
 }
}
```

Grafoak/Grafos

25

## Grafoen korritzeak

- Sakonerako korritzea:  
Depth First Traversal (DFT)
- Zabalerako korritzea:  
Breadth First Traversal (BFT)

# Zabalerako korritzea

- Erpin batetik hasten da
- Erpin horren auzokideak korritzen hasten da, maila berekoak korrituz aurrera egin baino lehen
- Behin maila berekoak bisitatu dituenean, horietatik lehena hartu eta bere auzokideak bisitatuko ditu, berriz ere zabaleran
- Eta horrela jarraitzen du errekurtsiboki, irsgarriak diren erpin guztiak behin bakarrik korritu arte

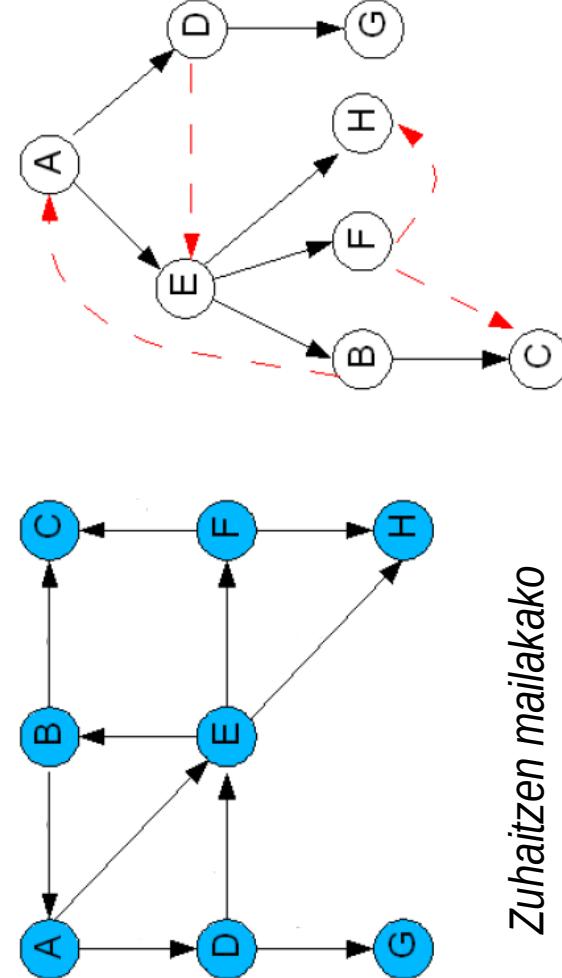
08/09/2022

Datu-Egiturak eta Algoritmoak

27

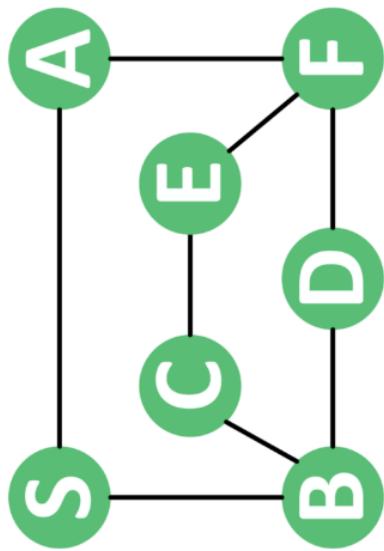
DEA

## Zabalerako ibilbidea



Zuhaitzen mailakako  
ibilbidearen tankerakoa

## Zabalerako korritzea. Adibidea

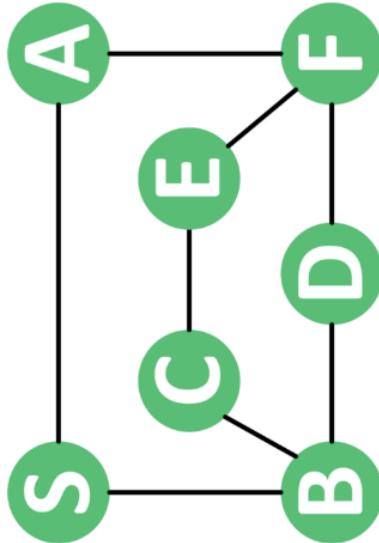


08/09/2022

Datu-Egiturak eta Algoritmoak

29

## Zabalerako korritzea. Adibidea



\* Auzokideen artean, ordena alfabetikoa jarraitzen dugu.

1. Stik hasita, bi auzokide: A eta B. A hartuko dugu\*
2. A bisitatu (bere auzokideak itxarote-ilaran jarriko ditugu)
3. B bisitatuko dugu (bere auzokideak itxarote-ilaran jarriko ditugu)
4. Ondoren F bisitatuko dugu, Aren auzokide bisitatu gabea. D eta E ilaran sartuko dira
5. C eta D bisitatuko ditugu
6. Bukatzeko E
7. Ilaran ez dago erpinlik bisitatu gabe. Bukatu da.

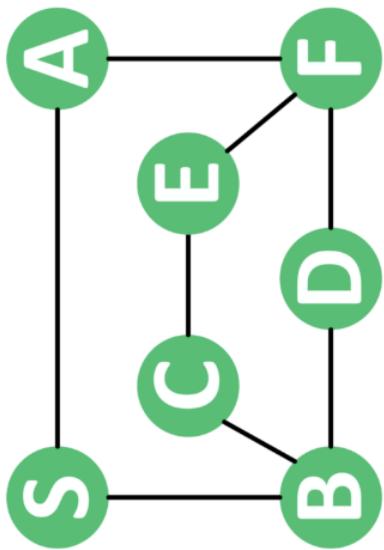
08/09/2022

Datu-Egiturak eta Algoritmoak

30

# Sakonerako korritzea.

## Adibidea



Zabalerako korritzea:  
 $\{S, A, B, F, C, D, E\}$

08/09/2022

Datu-Egiturak eta Algoritmoak

31

DEA

## Zabalerako ibilbidea

```
proc RECORRIDOENANCHURA ($G = (N, A)$)
 for cada $v \in N$ loop marca(v) \leftarrow falso end loop
 for cada $v \in N$ loop
 if \neg marca(v) then MARCAAANCHO(v)
 end loop

 proc MARCAAANCHO (v)
 $C \leftarrow$ new Cola
 marca(v) \leftarrow verdadero
 $C.insert(v)$
 while $\neg C.is_empty()$ loop
 $u \leftarrow C.remove_first()$
 for cada $w \in N$ adyacente de u loop
 if \neg marca(w) then
 marca(w) \leftarrow verdadero
 $C.insert(w)$
 end while
 end proc
```

## Zabalerako ibilbidearen algoritmoa Java

```

LinkedList<Erpina> zabalerakoKorritzea(Erpina abiapuntua) {
 LinkedList<Erpina> bisitatuak = new LinkedList<Erpina>();
 Queue<Erpina> itxaroteIlara = new LinkedList<Erpina>();
 itxaroteIlara.add(abiapuntua);
 bisitatuak.add(abiapuntua);
 while (!itxaroteIlara.isEmpty()) {
 Erpina erpina = itxaroteIlara.poll();
 for (Erpina v : this.auzokideak(erpina)) {
 if (!bisitatuak.contains(v)) {
 bisitatuak.add(v);
 itxaroteIlara.add(v);
 }
 }
 return bisitatuak;
 }
}
```

08/09/2022

Data-Egiturak eta Algoritmoak

33

## Afaria, biltzarrean (1)

```

func CONVENCIÓN ($G = (N, A)$) return boolean
 es_posible $\leftarrow true$ {valor inicial}
 for cada $v \in N$ loop marca(v) $\leftarrow false$ end for
 for cada $v \in N$ loop
 if not marca(v) then MARCA_COMEDOR($v, true, es_posible$) end if
 if not es_posible then return $false$ end if
 end for
 return $true$
```

DEA

**Adibidea:** **afaria, biltzarrean.** Biltzar batean, afari bat antolatu behar dute, eta bijangela dituzte horretarako. Partehartzaile asko haserretuta daude, eta ez dute jangela berean egon nahi. Posible al da partehartzaileak bijangela horietan banatzea, haserre daudenak jangela desberdinetan jarrita?

## Afaria, biltzarrean (2)

```

proc MARCA_COMEDOR(v , opción, resultado)
 $C \leftarrow$ Cola_vacia()
 marca(v) \leftarrow true
 comedor(v) \leftarrow opción
 C.anadir(v)
 while not C.vacia() loop
 $u \leftarrow$ C.retirar-primer()
 for cada w adyacente de u loop
 if not marca(w) then
 marca(w) \leftarrow true
 comedor(w) \leftarrow not comedor(u) {llevar a w a otro comedor}
 C.anadir(w)
 else { w está marcado, y por tanto asignado a un comedor.}
 {Si es el mismo que el de u entonces no es posible repartirlos}
 if comedor(u) = comedor(w) then
 resultado \leftarrow false
 return

```

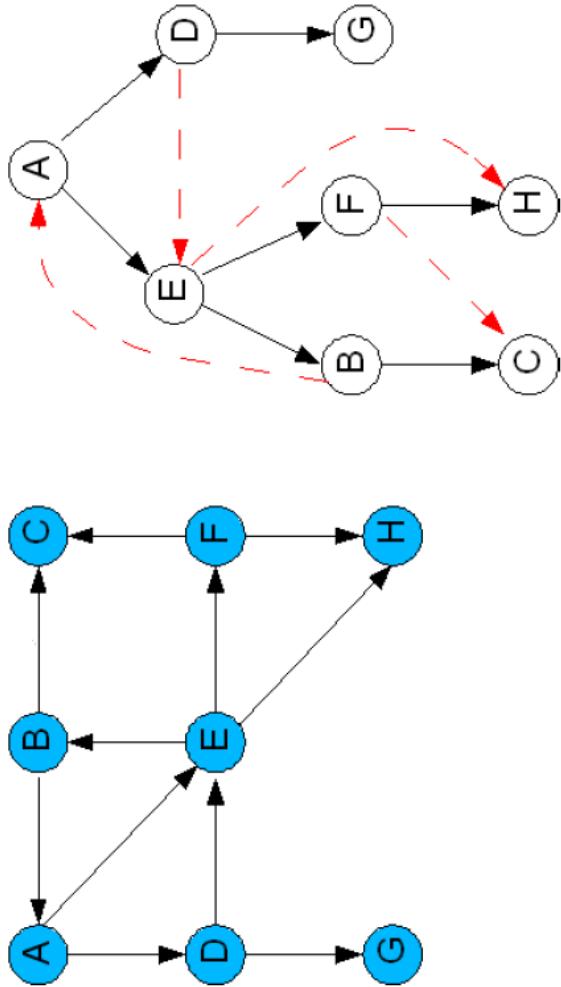
Grafoak/Grafos

35

## Sakonerako korritzea

- Erpin batetik hasten da
- Erpin horren auzokideak korritzen hasten da, baina maila bereean aritu ordez, horietako bat hartu eta bidean aurrera egiten du *sakonerantz*
- Sakoneran azkenera iristen denean, atzera egiten du
- Eta horrela jarraitzen du errekurtsiboki, irisgarriak diren erpin guztiak behin bakarrik korritu arte

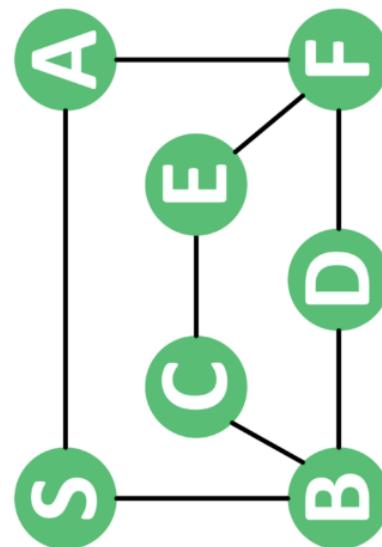
## Sakonerako ibilbidea



Grafoak/Grafos

37

## Sakonerako korritzea. Adibidea

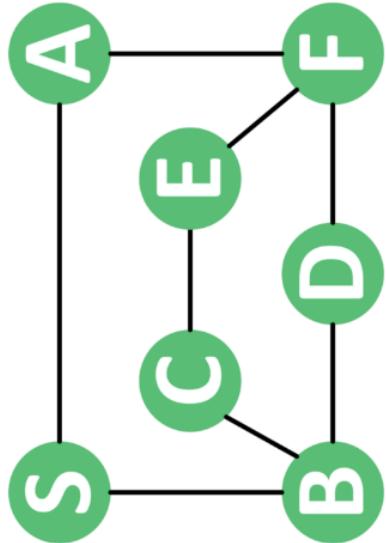


08/09/2022

Data-Egiturak eta Algoritmoak

38

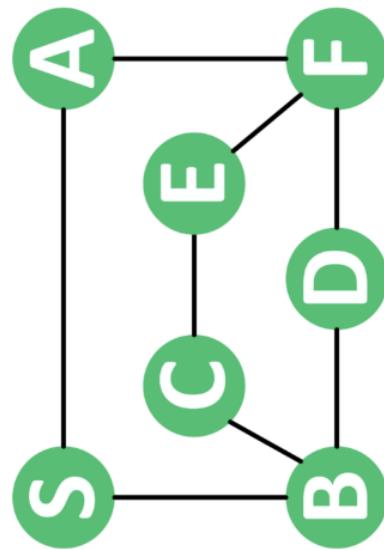
# Sakonerako korritzea. Adibidea



\* Auzokideen artean, ordena alfabetikoa jarraitzen dugu.

1. Stik hasita, bi auzokide: A eta B. A hartuko dugu\*
2. Ak bisitatu gabeko auzokide bakarra duenez, Ftik segituko dugu
3. Fk bisitatu gabeko bi auzokide: D eta E. D hartuko dugu\*
4. Dk auzokide bakarra du: B
5. Bk auzokide bakarra du: C
6. Ck auzokide bakarra du: E
7. Ek ez du bisitatu gabeko auzokiderik, ez Ck, ez Bk, ez Dk, ez Fk, ez Ak, ez Sk
8. Bilkatu da.

# Sakonerako korritzea. Adibidea



Sakonerako korritzea:  
 $\{S, A, F, D, B, C, E\}$

# Sakonerako ibilbidea

```

proc RECORRIDOENPROFUNDIDAD ($G = (N, A)$)
 for cada $v \in N$ loop marca(v) \leftarrow falso end loop
 for cada $v \in N$ loop
 if \neg marca(v) then MARCAPROF(v)

```

```

proc MARCAPROF (v)
 marca(v) \leftarrow verdadero
 for cada $w \in N$ adyacente de v loop
 if \neg marca(w) then MARCAPROF(w)

```

Ariketa: 11. gărdeneikiko algoritmoan zein  
aldaketa SOTIL egingo zenuke, sakonerako  
ibilbide hau lortzeko

Grafoak/Grafos

4.1

## Sakonerako ibilbidea (eskema orokorra)

```

proc MARCA_PROF_GEN(v)
 [Procesar v en primera visita]
 [Como en preorden]
 marca(v) \leftarrow cierto
 for cada $w \in N$ adyacente de v loop
 [Procesar arista (v, w)]
 if \neg marca(w) then
 [Procesar arista (v, w) del árbol]
 MARCA_PROF_GEN(w)
 [Procesar v al regreso de procesar w]
 [Como en inorden]
 else
 [Procesar arista (v, w). NO es del árbol]
 end for
 [Procesar v al abandonarlo]
 [Como en postorden]

```

Grafoak/Grafos

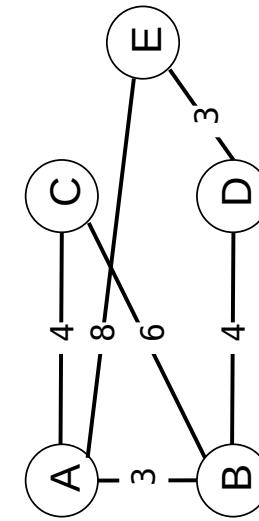
4.2

## Zabalerako eta sakonerako ibilbideen analisia

- Izanik  $n$  grafoko adabegien kopurua eta  $a$  arkuen kopurua:
  - $O(n+a)$  baldin eta grafoa auzokidetasun listekin adierazten bada.
  - $O(n^2)$  baldin eta grafoa auzokidetasun matrizeekin adierazten bada.

## Bide laburrena (shortest path)

- Grafo ponderatua
- Bideak bilatu:
  - Bilatu bide laburrena A-tik D-ra



# Biderik motzena

- Grafoen teorian, **biderik motzenaren problema** (**shortest path problem (SPP)**) oso ezaguna da.
- **Dijkstraaren algorimoak** problema honen ebazpena planteatzen du.
  - Algoritmo honek, abiapuntu gisako erpin bat hartuta, gainontzeko erpin irsgarrietarako biderik motzena aurkitzea du helburu.

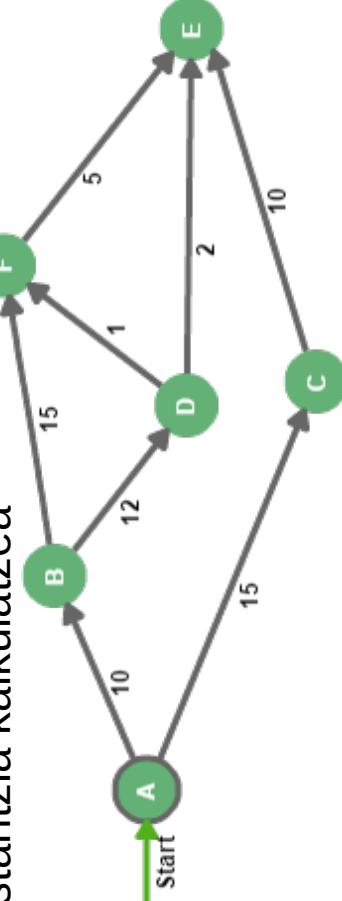
08/09/2022

Datu-Egiturak eta Algoritmoak

45

# Dijkstraaren biderik motzenaren algoritmoa

- Grafo zuzendu ponderatu bat emanda, ideia da hasierako erpin batetik abiatzea (irudiko A), eta gainontzeko erpinetarako biderik motzena eta distantzia kalkulatzea



Iturria: <https://www.baeldung.com/java-dijkstra>

08/09/2022

Datu-Egiturak eta Algoritmoak

46

## Dijkstraren biderik motzenaren algoritmoa (II)

- Bi erpin-multzoa erabiltzen ditu algoritmoak: ebatziak eta ebatzi gabekoak.
- Ebatzien multzoen dauden erpinetarako badakigu zein den abiapuntutik erpin horietara iristeko distantzia minimoa.
- Ebatzi gabekoena multzoan daude irisgarriak diren erpinak, baina oraindik ez dakigu zein den distantzia minimoa abiapuntutik.

08/09/2022

Datu-Egiturak eta Algoritmoak

47

## Dijkstraren biderik motzenaren algoritmoa (III), **urratsez urreats**

- Hasieratu abiapuntuErpinerako distantzia (= 0)
- Beste distantzia guztiei balio maximoa (*infinitua*) esleitu
- abiapuntuErpina ebatziGabeak multzora gehitu
- ebatziGabeak multzoa hutsik ez dagoen bitartean:
  - ebatziGabeak multzotik aukeratu ebaluaziorako erpin bat. Erpin hori izango da abiapuntutik distantzia txikienera dagoena
  - Kalkulatu distantzia berriak auzokideetara, ebaluazio bakoitzeko distantzia txikiiena hartuta
  - Oraindik ebatzi gabe dauden auzokideak ebatziGabeak multzora erantsi

08/09/2022

Datu-Egiturak eta Algoritmoak

48

## Dijkstraren biderik motzenaren algoritmoa (IV): bi fase

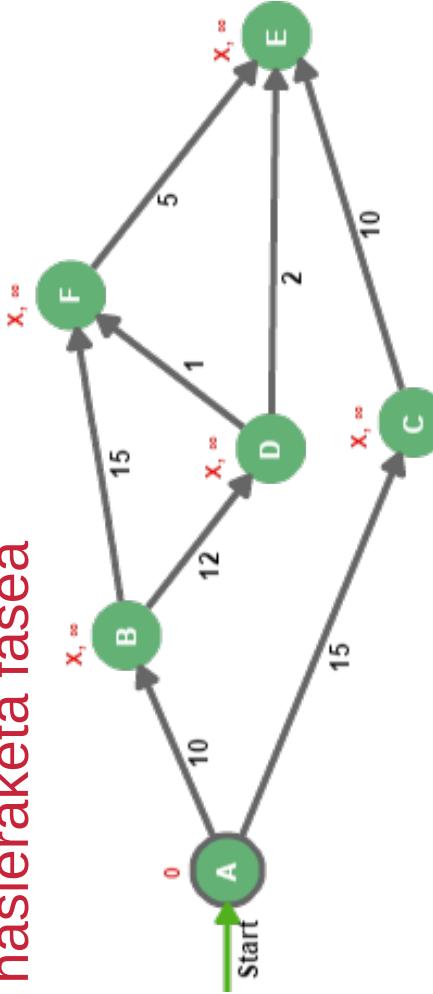
- Aurreko urratsak bi fasetan bereiz daitezke:
  - Hasieraketa eta ebaluazioa
  - **Hasieraketa:**
    - Grafoko bideak esploratu aurretik, erpin guztiak distantzia infinituarekin eta bide ezezagunarekin hasieratzen dira (abiapuntua izan ezik)
    - Abiapuntua 0 distantziarekin hasieratzen da

08/09/2022

Datu-Egiturak eta Algoritmoak

49

## Dijkstraren biderik motzenaren algoritmoa (V): hasieraketa fasea



- Hasieraketa bukatzeko, A erpina ebatziGabeak multzoan sartuko dugu, eta erpin horretatik hasiko da ebaluazio-fasea.
- ebatziak multzoa hutsik dago oraindik.

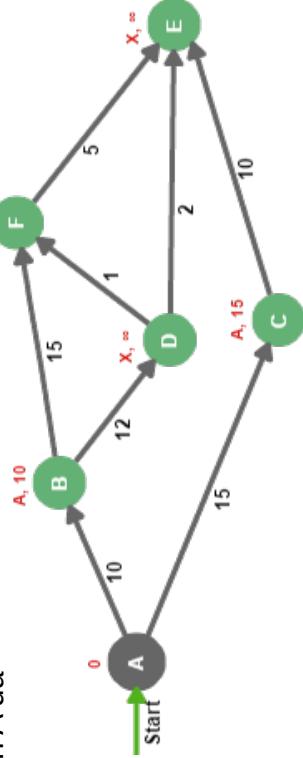
08/09/2022

Datu-Egiturak eta Algoritmoak

50

## Dijkstraren biderik motzenaren algoritmoa (V): ebaluazio fasea

- ebatziGabeakoak multzotik distantzia txikienekoa hartu, eta ebatzi gabeko auzokideak ebaluatuko ditugu:
  - Ebaluazio-erpineko distantzia kalkulatzeko, aurrekariaren distantziari arkuaren pisua gehitzen zaio, eta konparatu egiten da ebaluazio-erpinak lehendik zeukan distantziarekin. Adb.:  $0 + 10 < \infty$
  - Irudian: B eta C erpinei distantzia berriak ezartzen zaizkie, eta aurrekaria orain A da



08/09/2022

Datu-Egiturak eta Algoritmoak

51

## Dijkstraren biderik motzenaren algoritmoa (VI): ebaluazio fasea

- Irudiko adibidean, behin B eta C erpinen distantzia motzenak kalkulatuta:

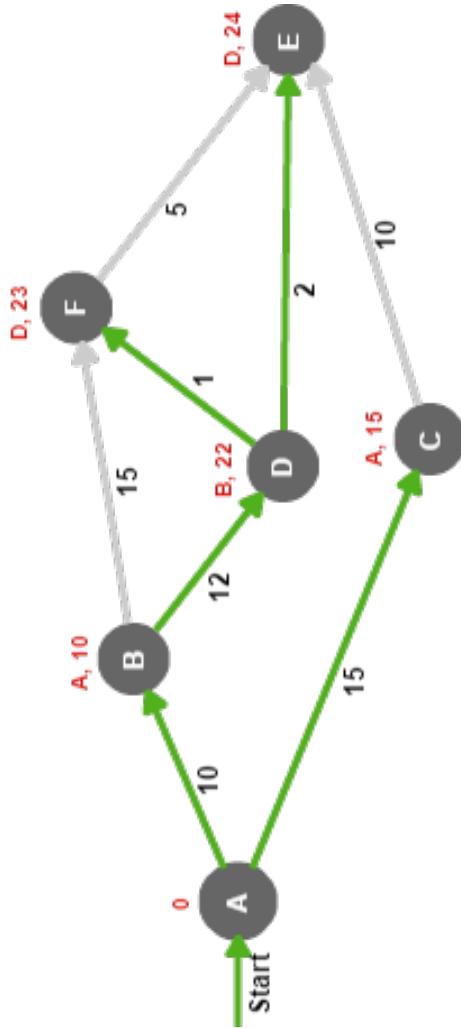
- A erpina ebatziak multzoan eransten da (dagoeneko ebaluatura dago, bere auzokide guztien distantzia motzenak kalkulatu dira eta)
- B eta C erpinak ebatziGabeakoak multzoan eransten dira, irisgarriak baitira, baina ebaluatu egin behar dira
- Bi erpin ditugu, hortaz, ebatziGabeakoak multzoan. Horietatik distantzia txikienekoa hartuko dugu (B erpina), eta horrela jokatuko dugu erpin irisgarri guztiak ebatzi arte

08/09/2022

Datu-Egiturak eta Algoritmoak

52

## Dijkstraren biderik motzenaren algoritmoa (VII): ebaluazio fasea



08/09/2022

Datu-Egiturak eta Algoritmoak

53

## Dijkstraren bide motzena algoritmoa, Java implementatua

Erpinak klassea:

```
public class Erpina {
 private String etiketa;
 private List<Erpina> bideMotzena = new LinkedList<>(); // Abiapuntutik bide motzena
 private Integer distantzia = Integer.MAX_VALUE; // bide motzenaren distantzia
 private Map<Erpina, Integer> erpinAuzokideak = new HashMap<>();
 // auzokideen hash taula
 // item bakoitza: erpin auzokidea eta arkuaren pisua

 public void erantsiAuzokidea(Erpina auzokidea, int distantzia) {
 erpinAuzokideak.put(auzokidea, distantzia);
 }
 // getterrak eta setterrak
}
```

08/09/2022

Datu-Egiturak eta Algoritmoak

54

### Grafoa klasea:

```

public class Grafoa {

 private Set<Erpina> erpinak = new HashSet<>(); // grafoa erpinen multzoa da

 public void erantsiErpina(Erpina u) {
 erpinak.add(u);
 }

 public static Grafoa kalkulatuBideMotzenaAbiapuntutik(Grafoa grafoa, Erpina abiapuntua) {
 abiapuntua.setDistantzia(0);
 Set<Erpina> erpinEbatziak = new HashSet<>();
 Set<Erpina> erpinEbatziGabeak = new HashSet<>();
 erpinEbatziGabeak.add(abiapuntua);

 while (erpinEbatziGabeak.size() != 0) {
 Erpina unekoErpina = LortuDistantziaTxikienekoErpina(erpinEbatziGabeak);
 erpinEbatziGabeak.remove(unekoErpina);
 for (Map.Entry<Erpina, Integer> auzokidea: unekoErpina.getErpinAuzokideak().entrySet()) {
 Erpina erpinAuzoa = auzokidea.getKey();
 Integer arkuPisua = auzokidea.getValue();
 if (!terpinEbatziak.contains(terpinAuzoa)) {
 KalkulatuBistantziaMinimoa(terpinAuzoa, arkuPisua, unekoErpina);
 erpinEbatziGabeak.add(terpinAuzoa);
 }
 }
 erpinEbatziak.add(unekoErpina);
 }
 return grafoa;
 }
}

```

08/09/2022

Datu-Egiturak eta Algoritmoak

55

```

// erpinEbatziGabeak multzotik distantzia txikiena duen erpina itzultzen du
private static Erpina LortuDistantziaTxikienekoErpina(Set<Erpina> erpinEbatziGabeak) {
 Erpina distantziaTxikienekoErpina = null;
 int distantziaTxikiena = Integer.MAX_VALUE;
 for (Erpina erpina: erpinEbatziGabeak) {
 int erpinDistantzia = erpina.getDistantzia();
 if (erpinDistantzia < distantziaTxikiena) {
 distantziaTxikiena = erpinDistantzia;
 distantziaTxikienekoErpina = erpina;
 }
 }
 return distantziaTxikienekoErpina;
}

```

08/09/2022

Datu-Egiturak eta Algoritmoak

56

```

 // abiapuntuErpina-ren auzokidea da ebaluazioErpina, eta biak lotzen dituen arkuaren pisua da
 // ebaluazioErpina-ren bideMotzena eta distantzia atributuak eguneratzen ditu:
 // horretarako abiapuntuErpina-tik barrena doan bidearen distantzia uneko distantziarekin
 // komparatzen du.

private static void kalkuluDistantziaMinimoa(Erpina ebaluazioErpina,
 Integer arkuPisua,
 Erpina abiapuntuErpina) {
 Integer abiapuntukoDistantzia = abiapuntuErpina.getDistantzia();
 if (abiapuntukoDistantzia + arkuPisua < ebaluazioErpina.getDistantzia()) {
 ebaluazioErpina.setDistantzia(abiapuntukoDistantzia + arkuPisua);

 LinkedList<Erpina> bideMotzena = new
 LinkedList<>(abiapuntuErpina.getBidemotzena());
 bideMotzena.add(abiapuntuErpina);
 ebaluazioErpina.setBideMotzena(bideMotzena);
 }
}

```

08/09/2022

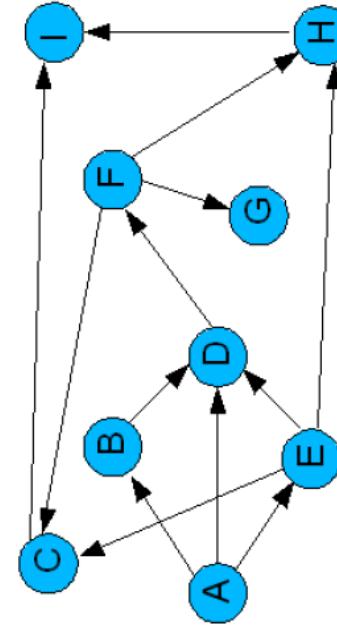
Datu-Egiturak eta Algoritmoak

57

DEA

## Ordenazio topologikoa

$G = (N, A)$  zuzendutako grafo ez-zikliko baten ordenazio topologikoak berekin dakin grafoko adabegien lista bat, non  $A$ -ko  $(u, v)$  arku bakoitzerako u adabegia  $v$  adabegiaren aurretik agertuko den emaitzako listan.



Adibideko grafoaren zenbait ordenazio topologiko:

A B E D F C H I G  
A E B D F G H C I

# Ordenazio topologikoa

```

func ORDENACIÓN_TOPOLOGICA (G = (N, A)) return Lista_de_nodos
 L ← new Lista
 for cada v ∈ N loop marca(v) ← falso end loop
 for cada v ∈ N loop
 if ¬ marca(v) then ORDENTOPO(v, L)
 end for
 return L

proc ORDENTOPO (v, L)
 marca(v) ← verdadero
 for cada w ∈ N adyacente de v loop
 if ¬ marca(w) then ORDENTOPO(w, L)
 end for
 L.insert_first(v)

```

Grafoak/Grafos

59

## Javako liburutegiak grafoetarako

- Javak ez dauka besterik ezeko implementaziorik *Graph* datu-egiturarako.
- Hala ere, badira kode irekiko liburutegiak, grafoen implementazioak eskaintzen dituztenak eta oso erabilgarriak izan daitezkeenak
- Adibidez:
  - JGraphT (<https://jgrapht.org/>)

## Irakurgaiak

- Barne-txostena:
  - “Recorridos de grafos: Teoría y aplicaciones”  
Jesús Bermúdez de Andrés
- [Lewis, Chase 2010]
  - 13. kapitula

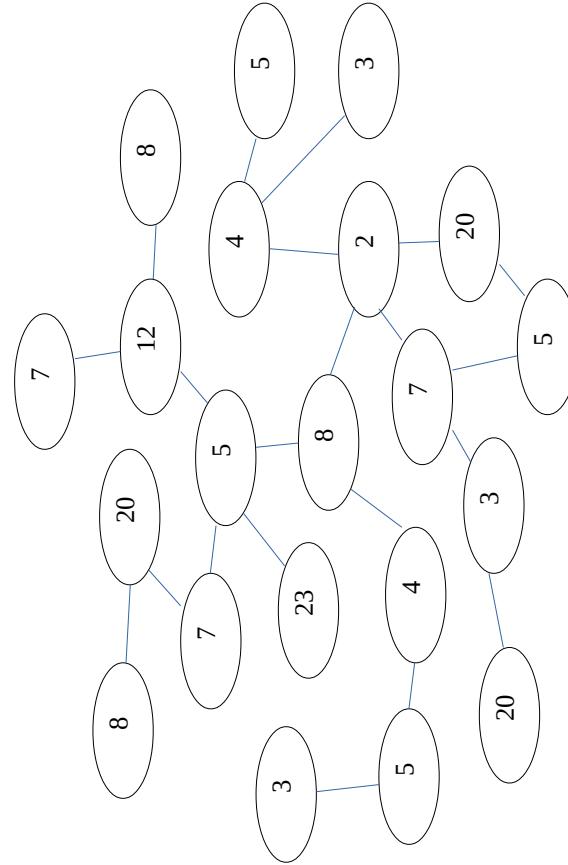
#### ARIKETA 4.- Adabegi baten dentsitatea baino altuagoko lortu (1,5 puntu)

**Grafo bat emanda**, auzokidetasun zerrenden bidez, algoritmo bat egin nahi da aurkitzeko, adabegi batetik abiatuta, D balioa baino dentsitate altuagoa edo berdina duen adabegi hurbilena aurkitzeko.

Adabegi bakoitzak int motako balio bat izango du (ikus azpian values izeneko arraya).

Adabegi baten dentsitatea bere balioa eta bere auzokideen balioen batura izango da. Hau da, i adabegianen dentsitatea honela lortzen da:

$$dentsitate(i) = V(i) + \sum_{j \in \text{adj } i} V(j)$$



```
public class GraphAL {
 protected int numVertices; // number of vertices in the graph
 protected LinkedList<Integer>[] adjList; // adjacency list
 protected int[] values; // values of nodes

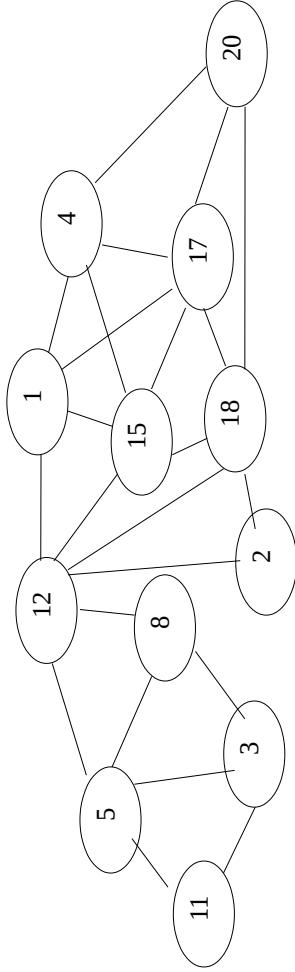
 public int lortuDentsitateAltuagokoAdabegia(int x, int d)
 // Aurrebaldintza: x grafoloko adabegi baten indizea da
 // Postbaldintza: emaitza x-ekik hurbilen dagoen eta d balioa baino dentsitate
 // berdina edo altuagoa duen adabegiaren indizea da,
 // edo -1, horrelako adabegirik ez balego
}
```

Ondokoa eskatzen da:

- `lortuDentsitateAltuagokoAdabegia()` metodoa implementatzea eskatzen da.
  - Algoritmoaren kostua kalkulatu, **modu arrazoiutan**

#### 4. Bidea bilatu (2,5 puntu)

Ondoko irudiak grafo bat adierazten du:



Balioen zerrenda bat emanda, zerrrendako elementuak konektatzen duen bidea dagoen jakin nahi da. Hau da, zerrrendan jarraian doazen elementuen ( $x, y$ ) bikote bakoitzeko, grafoak arku bat du  $x$ -etik  $y$ -raino.

Adibidez:

- <12, 1, 4, 17, 18, 15> zerrrendarekin emaitza true izango da, bide hori grafoan baitago, eta arku hauek daudelako: (12, 1), (1, 4), (4, 17), (17, 18), eta (18, 15)
- <12, 1, 20, 17, 18, 15> zerrrendarekin emaitza false izango da, ezin delako 1-etik 20-ra joan (ez dago arku zuzena)

```
public class Grafo
{
 protected int numVertices; // number of vertices in the graph
 protected int[][] adjMatrix; // adjacency matrix

 public boolean bideaDago(ArrayList<Integer> lista)
}
```

Hau eskatzen da:

- Algoritmoa implementatu
- Algoritmoaren kostua kalkulatu, modu arrazoituan.

#### **ARIKETA 4.- Konpilazioa (1,5 puntu)**

Programazio-lengoaia guzietan programen arteko erlazioak egoten dira. Era horretan, PR1 programa batek PR2, PR3 programak erabiltzen ditu, eta hauek PR4, PR5 ... programak erabil ditzakete. Hainbeste erlazioarekin, konpiladore batek jakin behar du ea, programa bat emanda, berak erabiltzen dituen programa guziak definituta dauden (zuzenean eta zeharka ere).

Programen arteko erlazioak gordetzeako hash taula bat erabiliko dugu, gakoa programaren izena delarik, eta datua programa horrek erabiltzen dituen programen zerrenda.

**hash taula**

| Gakoa | Datuak (erabilitako programen zerrenda) |
|-------|-----------------------------------------|
| PR1   | PR2, PR3                                |
| PR2   | PR5, PR3                                |
| PR15  | PR5, PR14                               |
| PR3   | PR5                                     |
| PR5   | PR2                                     |
| PR14  | PR25                                    |

Adibidez, PR1 programa zuzena da, taulan definituta dagoelako eta, gainera, berak erabiltzen dituen PR2, PR3 eta PR5 programak ere definituta daudelako (zuzenean edo beste programa batzuen bitartez). Era berean, PR15 ez da zuzena, zeharka PR25 erabiltzen duelako, eta programa hori ez dago definituta hash taulan.

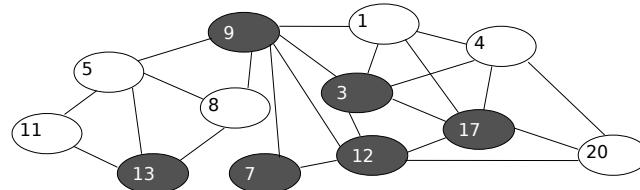
*zuzenaDa(programa) implementatu* behar da, programa hori ondo konpilatu daitekeen jakiteko, eta bere kostua kalkulatu.

```
public class Konpiladorea {
 HashMap<String, ArrayList<String>> mendekotasunak;

 public boolean zuzenaDa(String prog)
 // Postbaldintza: emaitza true "prog" programak erabiltzen dituen programa GUZTIAK
 // definituta baldin badaue, eta false bestela
}
```

### 3. Jokoak (2,5 puntu)

Ondoko irudiak joko bat adierazten du.



Jokoaren helburua hasierako adabegi batetik bukaerako adabegi batera iristea badagoen asmatzea da. Horretarako jokoaren arau nagusia errespetatu behar da:

- Kolore bateko lauki batetik bakarrik beste koloreetako laukietara bakarrik pasa daiteke (hau da, lauki beltzetik zurira, edo zuritik beltzera).

Algoritmo bat egin nahi dugu, jokoan dauden bi lauki emanda, bi elementu horiek konektatzen dituen bide minimoa emango diguna. Algoritmoak hau bueltatuko du:

- Zerrenda hutsa, laukien arteko konexiorik ez dagoenean
- Luzera minimoko bide bat baino gehiago balego, orduan edozein bueltatu daiteke

Adibidez, bilatuBidea("11", "20") deiak  $\langle 11, 13, 8, 9, 1, 3, 4, 17, 20 \rangle$  zerrenda bueltatuko luke (egon daitezke luzera horretako edo gehiagoko beste bide batzuk).

```
public class Laukia {
 String kolorea; // "zuria" edo "beltza"
 int balioa;
}

public class GrafoJokoak
{
 private HashMap<Integer, ArrayList<Laukia>> ondokoak;
 // gakoa: laukiko balioa
 // balioa: lauki horrekin konektatuta dauden laukiak

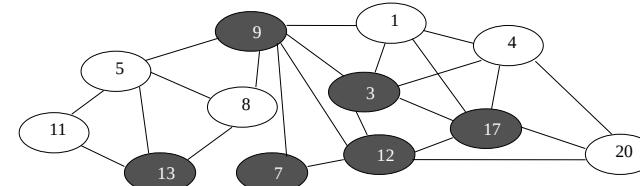
 public SimpleLinkedList<Laukia> bilatuBidea(Laukia hasiera, Laukia bukaera)
}
}
```

Adibidez, aurreko irudiko grafoa era honetan adieraziko litzateke hash taula baten bidez:

| clave | valor                          |
|-------|--------------------------------|
| 11    | $\langle 5, 13 \rangle$        |
| 5     | $\langle 11, 13, 8, 9 \rangle$ |
| 13    | $\langle 11, 5, 8 \rangle$     |
| 7     | $\langle 9, 12 \rangle$        |

### 3. Jokoak (1,5 puntu)

Ondoko irudiak joko bat adierazten du.



Jokoaren helburua hasierako adabegi batetik bukaerako adabegi batera iristea badagoen asmatzea da. Horretarako jokoaren arau nagusia errespetatu behar da:

- Kolore bateko lauki batetik bakarrik beste koloreetako laukietara bakarrik pasa daiteke (hau da, lauki beltzetik zurira, edo zuritik beltzera).

Algoritmo bat egin nahi dugu, jokoan dauden bi lauki emanda, bi elementu horiek konektatzen dituen bide minimoa emango diguna. Algoritmoak hau bueltatuko du:

- Zerrenda hutsa, laukien arteko konexiorik ez dagoenean
- Luzera minimoko bide bat baino gehiago balego, orduan edozein bueltatu daiteke

Adibidez, bilatuBidea("11", "20") deiak  $\langle 11, 13, 8, 9, 1, 3, 4, 17, 20 \rangle$  zerrenda bueltatuko luke (egon daitezke luzera horretako edo gehiagoko beste bide batzuk).

```
public class Laukia {
 String kolorea; // "zuria" edo "beltza"
 int balioa;
}

public class GrafoJokoak
{
 private HashMap<Integer, ArrayList<Laukia>> ondokoak;
 // gakoa: laukiko balioa
 // balioa: lauki horrekin konektatuta dauden laukiak

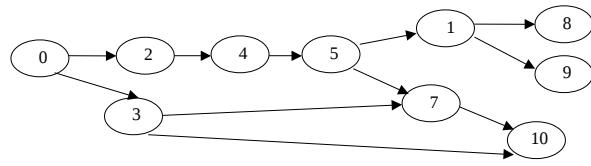
 public SimpleLinkedList<Laukia> bilatuBidea(Laukia hasiera, Laukia bukaera)
}
```

Adibidez, aurreko irudiko grafoa era honetan adieraziko litzateke hash taula baten bidez:

| clave | valor                          |
|-------|--------------------------------|
| 11    | $\langle 5, 13 \rangle$        |
| 5     | $\langle 11, 13, 8, 9 \rangle$ |
| 13    | $\langle 11, 5, 8 \rangle$     |
| 7     | $\langle 9, 12 \rangle$        |

#### 4. Ordenagailuen sarea (1,5 puntu)

Eman dezagun ordenagailu-sare bat dugula. X ordenagailu bakoitza bere auzoko Y ordenagailu bakoitzarekin komunika daitake euro 1eko prezioan.



Idatzi algoritmo bat, zeinak prezioa(1..n) taula kalkulatuko duen:  
prezioa(i) = 0 ordenagailuak i ordenagailuarekin konektatzeko ordaindu beharreko prezio minimoa izanik.

```

public class OrdenagailuenSarea {

 protected Boolean[][] adjMatrix; // adjacency matrix
 public int[] kostuakLortu() // Egin behar den metodoa
}

```

## 439 Knight Moves

A friend of you is doing research on the *Traveling Knight Problem (TKP)* where you are to find the shortest closed tour of knight moves that visits each square of a given set of  $n$  squares on a chessboard exactly once. He thinks that the most difficult part of the problem is determining the smallest number of knight moves between two given squares and that, once you have accomplished this, finding the tour would be easy.

Of course you know that it is vice versa. So you offer him to write a program that solves the "difficult" part.

Your job is to write a program that takes two squares  $a$  and  $b$  as input and then determines the number of knight moves on a shortest route from  $a$  to  $b$ .

### Input

The input file will contain one or more test cases. Each test case consists of one line containing two squares separated by one space. A square is a string consisting of a letter (a..h) representing the column and a digit (1..8) representing the row on the chessboard.

### Output

For each test case, print one line saying 'To get from  $xx$  to  $yy$  takes  $n$  knight moves.'

### Sample Input

```

e2 e4
a1 b2
b2 c3
a1 h8
a1 h7
h8 a1
b1 c3
f6 f6

```

### Sample Output

```

To get from e2 to e4 takes 2 knight moves.
To get from a1 to b2 takes 4 knight moves.
To get from b2 to c3 takes 2 knight moves.
To get from a1 to h8 takes 6 knight moves.
To get from a1 to h7 takes 5 knight moves.
To get from h8 to a1 takes 6 knight moves.
To get from b1 to c3 takes 1 knight moves.
To get from f6 to f6 takes 0 knight moves.

```



## Hash taulak



- **Zer dira?**
  - Zergatik dira garrantzitsuak
  - Zertarako erabiltzen dira
  - Ezaggarriak
- **Metodo nagusiak**
  - Hash taula itxia
  - Hash taula irekia



- **Gako** baten bidez bereizi daitezkeen elementuen multzo handiak **kokatu** eta **atzitzeko** beharra.
- Adibidez:
  - Hiztegi bateko definizioak
  - Datu pertsonalen erregistroak
  - Katalogo industrialetako elementuak: argitalpenak, mekanika, ...
  - Konpiladoreek kudeatzen dituzten sinboloak
  - Jokoetako egoerak
  - Datu-multzo handiak

3



## Hash taulak

Zergatik dira garrantzitsuak: Helburuak

Elementu talde batean, oinarritzko eragiketak era ergaginkorrenean exekutatzen dira:

- **Bilaketa**
- **Ezabaketa**
- **Txertaketa**

Hash taulen helburua eragiketa guztiak **ordena konstantean** egitea da  $O(1)$

4



## Hash Taulak

Zer dira? Zertarako erabiltzen dira?

- Datu egitura honek elementu multzo bat **indexatzen** du funtzio bat erabiliz (hash-funtzioa)

- **Array-a** da erabiltzen den datu egitura

5



## Hash Taulak : Adibideak

Nortasun Agiriak:

|            |            |             |
|------------|------------|-------------|
| 44.000.001 | ██████████ | 100 langile |
| 15.235.567 | ██████████ |             |
| 45.567.235 | ██████████ |             |
| 21.876.897 | ██████████ |             |
| ...        |            |             |

6



# Hash Taulak : Adibideak

Hiztegia:

|          |       |
|----------|-------|
| katua    | cat   |
|          |       |
| txakurra | dog   |
| etxea    | house |
| ...      |       |
| kaixo    | hello |

7

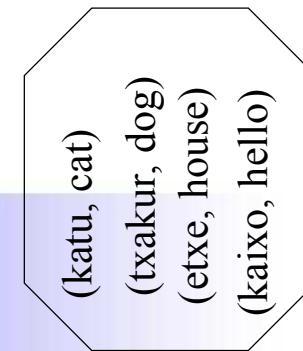
## Hash taulak

Zer dira? Zertarako erabiltzen dira?



## HASH TAULA

|    |                |
|----|----------------|
| 1  |                |
| 2  | (katu, cat)    |
| 3  |                |
| 4  |                |
| 5  | (txakur, dog)  |
| 6  |                |
| 7  | (etxe, house)  |
| 8  |                |
| 9  |                |
| 10 | (kaixo, hello) |
| 11 |                |
| 12 |                |



Adibidea:

$$\begin{aligned}f(\text{katu}) &= 2 \\ f(\text{txakur}) &= 5 \\ f(\text{etxe}) &= 7\end{aligned}$$



## Hash taulak hash funtzioa

- Elementuaren gakoa erabiliz array-aren indize bat itzultzen du

$f(gakoa) = \text{Array posizioa}$

- Ezauagarriak (desiragarriak)
  - **Elementu bakoitzaz posizio desberdinean**
  - **Konplexutasun ordena konstantea**
  -
- Hash funtzoak gako bat hartuta zenbaki positibo bat lortu behar du. Adibidez:
  - $\text{hash}(\text{"JAVA"}) = 10 + 1 + 23 + 1 = 35$
  - $\text{Hash}(\text{"15564465"}) = 36$

9



## Hash taulak: hash funtzioa Adibidea

- Hiztegi bateko 50.000 definizio gorde nahi ditugu. Definitzten den hitza gakoa da, eta definizioa balioa.
- 50.000 posizioko taula deifini genezake, eta hitz bakoitzari taulako indize bat esleitu
- Nola lotu hitz bat indize batekin?
  - Hash: Hitza -> indizea

10



## Hash taulak: hash funtzioa Adibidea. Lehen saiakera

- Demagun hitzak minuskuletan daudela
- Letra bakoitzaz zenbaki batekin kode daitetik a=1, b=2, ..., z=27), eta hitz bakoitzari bere letren kodeen batura esleituko diogu:  
 $\text{hash1}(\text{sei}) = 20 + 5 + 9 = 34$
- Arazoak:
  - $\text{hash1}(\text{baita}) = 2 + 1 + 9 + 21 + 1 = 34$
  - $\text{hash1}(\text{arpa}) = \text{hash1}(\text{para}) = \text{hash1}(\text{rapa})$
  - 10 letrako hitzetara mugatzen baldin bagara, indizerik handiena 270 da. Gorde nahi diren 50.000 hitzetatik oso urrun!
  - Hitz askok indize bera izango dute lotuta

11



## Hash taulak: hash funtzioa Adibidea. Bigarren saiakera

- Letra bakoitzaz zenbaki batekin kode daitetik a=1, b=2, ..., z=27), eta hitz bakoitzari kode hau esleituko diogu: “27 oinarrian dagokion zenbakia”

$$\text{hash2}(\text{baita}) = 2x27^4 + 1x27^3 + 9x27^2 + 21x27^1 + 1x27^0 = \\ 1062882 + 19683 + 6561 + 567 + 1 = 1089694$$

- Arazoa: 10 letretako hitz batek indize handiegia izango luke:  $27^9 > 7.000.000.000.000$

12



## Hash taulak: hash funtzioa Adibidea. Hirugarren saiakera

- 50.000 hitz gordetzeko, adibidez, 100.000 elementuko taula har dezakegu
- Zenbaki handi bat "konprimitu" dezakegu, 0..99.999 eremuauan egoteko, zatiketaren hondarra erabiliz:
  - $\text{hash2}(\text{clase}) \% 100.000 = 33.508$
- Ez dira arazo guztiak ekiditzten:

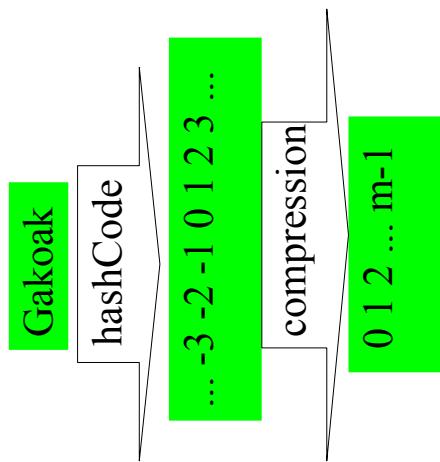
- $1733580 \% 100.000 = 2433580 \% 100.000$

13



## Hash taulak: hash funtzio baten forma tipikoa

- $\text{hashCode}(\text{baita}) = 2x27^4 + 1x27^3 + 9x27^2 + 21x27^1 + 1x27^0 = 1089694$
- $\text{compression}(1089694) = 1089694 \% 100.000 = 89.694$
- $\text{hash}(\text{baita}) = \text{compression}(\text{hashCode}(\text{baita}))$



14



## Hash taulak: hash funtzioa Adibidea.

- String motarako definituta dagoen hashCode funtzioa:

$$h(s) = \sum_{i=0}^{n-1} s[i] \cdot 31^{n-1-i}$$

- Adibidez: "Ea" eta "FB" kateek kode bera emango dute:

$$69 \times 31 + 97 = 70 \times 31 + 66$$

15

## Hash funtzio baten forma tipikoa

- a) Hash kodeketa

$$\text{hashKodea(baita)} = 2 \times 31^4 + 1 \times 31^3 + 9 \times 31^2 + 21 \times 31^1 + 1 \times 31^0 = \\ 1089694$$

- b) Trinkotzea

$$\text{trinkotzea(1089694)} = 1089694 \% 100.000 = 89.694$$

- c) Beraz:

$$\text{hash(baita)} = \text{trinkotzea(hashKodea(baita))}$$



## Hash taulak

Metodo nagusiak (interfazea)

### public interface Map<K, V>

| <b>Metodoak</b>                    | <b>Esanahia</b>                                                                                                                                                                                |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| V <b>put</b> (K key, V value)      | Gakoa eta balioa txertatzen ditu hash taulan. Gakoa badago, balio zaharra aldatzen du balio berriarengatik. Kasu honetan balio zaharra itzultzen du. Gakoa ez badago <b>null</b> itzultzen du. |
| V <b>get</b> (K key)               | Gakoari dagokion balioa itzultzen du, edo <b>null</b> ez badago.                                                                                                                               |
| V <b>remove</b> (K key)            | Gakoa eta bere balioa ezabatzen ditu. Gakoari dagokion balioa itzultzen du edo <b>null</b> existitzen ez bada.                                                                                 |
| void <b>clear</b> ()               | Elementu guztiek ezabatzen ditu.                                                                                                                                                               |
| boolean <b>isEmpty</b> ()          | True hutsa bada, edo false beste kasuan.                                                                                                                                                       |
| boolean <b>containsKey</b> (K key) | True gakoa taulan badago, false beste kasuan.                                                                                                                                                  |
| int <b>size</b> ()                 | Zerrendaren elementu kopurua itzultzen du                                                                                                                                                      |

Informazio osagarria:

[\*http://java.sun.com/j2se/1.5.0/docs/api/java/util/Map.html#getClass\(\)\*](http://java.sun.com/j2se/1.5.0/docs/api/java/util/Map.html#getClass())

17



## Hash taulak: interfazea

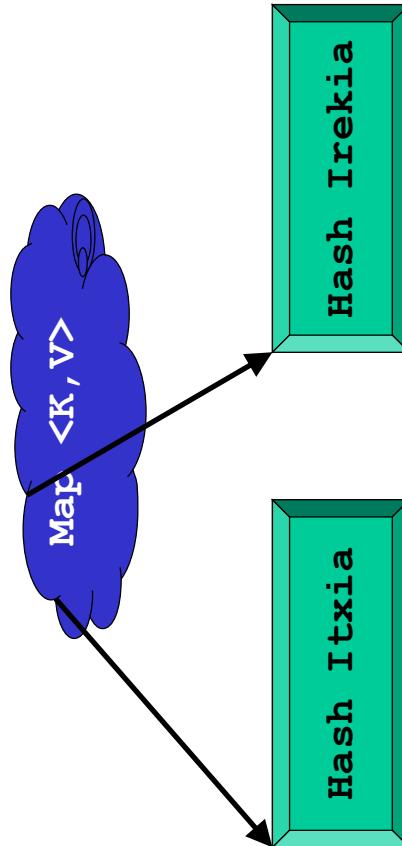
```
public interface Map <K,V> {
 public V put (K key, V value);
 public V get (K key);
 public V remove (K key);
 public void clear();
 public boolean isEmpty();
 public boolean containsKey(K key);
 public int size();
}
```

18



# Hash taulak

## Interfaze bat, implementazio desberdinak



19



## Hash taulak: Hash itxia

- Hash taula bat itxia da, onartzen ditueng gehievezko elementu kopurua finkatuta dagoenean (MAXPOS)
- Eraginkortasuna bermatzeko, taula gorde nahi den elementu kopurua baino **%25 handiagoa izatea gomendatzen da** (**karga-faktorea < % 75 = 0.75**)

20



## Hash taulak

### Hash itxia

- Hash funtziaren emaitzarekin gakoari taulan dagokion posizioa lortzen da
  - indize=hash(gakoa) % MAXPOS
  - Baldin MAXPOS = 20
    - 15564465 gakoaren posizioa zein litzateke?
    - “JAVA” gakoaren posizioa zein litzateke?

21



## Hash taulak

### Hash itxia

- Zer gertatuko litzateke posizio hori beste gako **sinonimo** batek beteko balu (Kolisioa edo talka)? →
  - *Lehenbailen libre dagoen posizio bat aurkitu taulan!!*
  - *Esaterako: “CC” gakoa “ADA” gakoaren sinonimoa da ikusitako hash funtziorekin*

hash(“CC”) --> 3+3 = 6  
hash(“ADA”) --> 1 + 4 + 1 = 6

22



## Hash taula itxiak:

Talkak: Nola aurkitu posizio berria?

- Proba lineala: ondoren dagoen lehen posizio librea bilatu
    - **while (gakoa posizioa erabilta) do**  
**gakoa = (1 + gakoa) % MAXPOS**  
**Desabantaila: clusterrek sor daitezke**
    - Zenbakiz lehenaren proba: MAXPOS zatitzen EZ duen zenbakiz lehen bat aukeratu (MAXPOS = 15 → P = 13)  
**gakoa = (P + gakoa) % MAXPOS**
    - Elementu baten ezabaketak ez du esan nahi guztiz desagertu behar dela (non/Item)
- |          |                |
|----------|----------------|
| 0        |                |
| 1        | (katua,cat)    |
| 2        |                |
| 3        |                |
| 4        | (txakurra,dog) |
|          | (etxea,house)  |
|          | (kaixo,hello)  |
|          |                |
|          |                |
| MAXPOS-1 |                |



## Hash taula itxiak:

Talkak: Nola aurkitu posizio berria?

- *Proba koadratikoa: x-en ondoren libre dagoen posizioa sekuentzia honekin bilatzen du:*  
**x+1, x+2<sup>2</sup>, x+3<sup>2</sup>, x+4<sup>2</sup>, x+5<sup>2</sup>,...**
- Clusterrek ekiditu nahi ditu
- Dena dela, posizio batera doazten gakoek bide bera jarraitzen dute



## Hash taula itxiak: Talkak: Nola aurkitu posizio berria?

- *Hashing bikoitzan*:  $x$ -en ondoren libre dagoen lehen posizioa bilatuko du, gakoaren gaineko beste hash funtziotan kalkulatutako **Pauso** baten bidez:
- *Adibidez*:  $\text{Pauso} = P - (\text{gako \% } P)$ ,  $P$  zenbakik lehena eta arrayaren tamaina baino txikiagoa izanik
- *Pauso ezin da inoiz 0* izan
- *Arrayaren tamainak zenbakik lehena izan behar du*

25



## Hash taula itxiak: Proba linealaren implementazioa: datuak eta taula

```
class DataItem<K,V> {
 K key;
 // data item (key)
 V data;
 public DataItem(K k, V v) { // eraikitzailea
 key = k;
 data = v;
 }
} // end class DataItem
```

```
class HashTableItxian<K,V> implements Map<K,V> {
 private DataItem<K,V>[] hashArray; // hash-taula egitura
 private int arraySize;
 private DataItem<K,V> nonItem;
```

26



## Hash taula itxiak: Eraikitzalea eta hash funtzia

```
public HashTableItxia(int size) { // eraikitzalea
 arraySize = size;
 hashArray = new DataItem[arraySize];
 nonItem = new DataItem(null, null); // elementu ezabatua
}
```

```
private int hashFunc(K key) {
 return (key.hashCode() % arraySize);
// Objektu bakoitzak kode desberdin bat
// dauka, sistemak emandakoa. Erreferentziatik lortzen da
// Beste bat nahi izanez gero, birdefinitu behar dugu
}
```



## Hash taula itxiak: containsKey metodoa. Proba lineala

```
public boolean containsKey (K pkey) { // find item with key

 int hashVal = hashFunc(pkey); // hash the key

 while ((hashArray[hashVal] != null) &&
 (!hashArray[hashVal].key.equals(pkey))) { // not found
 hashVal++; // go to next cell
 hashVal = hashVal % arraySize; // wraparound if necessary
 }

 if (hashArray[hashVal] == null)
 return false; // ez dugu aurkitu
 else
 return true;
}
```



## Hash taula itxiak: remove metodoa. Proba lineala

```
public V remove (K pKey) { // delete a DataItem
 int hashVal = hashFunc(pKey); // hash the key

 while ((hashArray[hashVal] != null) &&
 (!hashArray[hashVal].key.equals(pKey))) {
 hashVal++; // proba lineala
 hashVal = hashVal % arraySize;
 }

 if ((hashArray[hashVal] == null))
 return null;
 else {
 V temp = hashArray[hashVal].data; // save item
 hashArray[hashVal] = nonItem; // delete item
 return temp; }
}
```

29



## Hash taula itxiak: put metodoa. Proba lineala

```
public V put (K pKey, V pData) { // insert a DataItem
 // (assumes table not full)

 int hashVal = hashFunc(pKey);
 while((hashArray[hashVal] != null) &&
 (!hashArray[hashVal].equals(nonItem)) &&
 (!hashArray[hashVal].key.equals(pKey))) { // proba lineala
 hashVal++;
 hashVal = hashVal % arraySize;
 }
 if (hashArray[hashVal] == null){
 // gakoa ez dago: txertatu
 hashArray[hashVal] = new DataItem(pKey, pData);
 return null;
 }
 else
```

30



## Hash taula itxiak: put metodoaren jarraipena

```
else if (hashArray[hashVal].equals(nonItem)) { // aurerrago begiratu
 int reserva = hashVal;
 hashVal = (hashVal + 1) % arraySize;
 while (hashArray[hashVal] != null &&
 !hashArray[hashVal].key.equals(pKey) && hashVal != reserva) {
 hashVal++;
 hashVal = hashVal % arraySize;
 }
 if ((hashArray[hashVal] == null) || hashVal == reserva) {
 // gakoa ez dago: txertatu
 hashArray[reserva] = new DataItem(pKey, pData);
 return null;
 }
 else { // (hashArray[hashVal].key.equals(pKey))
 // aldatu elementu hau
 V temp = hashArray[hashVal].data;
 hashArray[hashVal].data=pData;
 return temp;
 }
} // (hashArray[hashVal].key.equals(pKey))
```



## Hash taula itxiak: put metodoaren jarraipena

```
else { // (hashArray[hashVal].key.equals(pKey))
 // aldatu elementu hau
 V temp = hashArray[hashVal].data;
 hashArray[hashVal].data=pData;
 return temp;
}
} // end put()
```



## Hash taula itxiaren adibidea

*th = new HashTable<Integer, String>(10)*

| Indizea          | DataItem |      |
|------------------|----------|------|
|                  | key      | data |
| 0                |          |      |
| 1                |          |      |
| 2                |          |      |
| 3                |          |      |
| 4                |          |      |
| 5                |          |      |
| 6                |          |      |
| 7                |          |      |
| 8                |          |      |
| 9                |          |      |
| <b>10 MAXPOS</b> |          | 33   |

A grey arrow points upwards from the value 33 at index 9 to the label **MAXPOS** at index 10.



## Hash taula itxiaren adibidea

Gakoek lau digitu osoak dituztela suposatuko dugu (dddd):  
HASH ('dddd') → d+d+d+d  
eta kolisioak proba linealaz ebazten direla.

Ondorengo eragiketa egin ondoren hash taularen egoera zein den adierazi:

```
th.put (new Integer(1237), "ADA");
th.put (new Integer(2237), "C");
th.put (new Integer(0111), "JAVA");
th.put (new Integer(2237),"C++");
th.remove (new Integer(0111));
th.put (new Integer(1111), "C#");
th.remove (new Integer(1237));
```



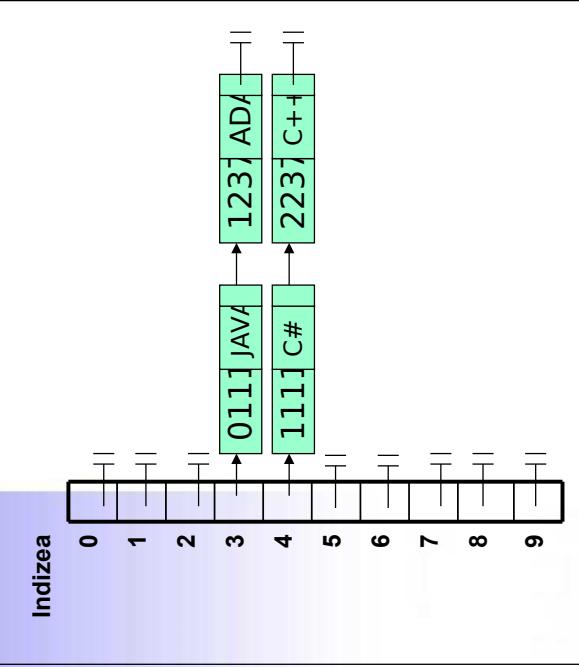
- **Ez dago** muga logikorik elementuak txertatzeko
- Hash funtzioa erabiliz gako bati dagokion posizioa lortzen da
- Posizio bereko sinonimoak zerrenda estekatu batean gordetzen dira
- Kolisioen kudeaketa ez da beharrezkoa

35



## Hash taula irekiaren adibidea

```
Map th = new
 HashTaularekia<Integer,
String>(10);
th.put(1237, "ADA");
th.put(2237, "C");
th.put(0111, "JAVA");
th.put(2237, "C++");
th.put(1111, "C#");
```



36



## Hash taula irekiak: Implementazioa: datuak eta taula

```
class DataItem<K,V> {
 public K key; // data item (key)
 public V data;
 public DataItem(K pKey, V pData) { // eraikitzaitzailea
 key=pKey;
 data=pData; }
 } // end class DataItem
```

```
class HashTableIrekia<K,V> implements Map<K,V> {
 LinkedList<DataItem<K,V>>[] hashArray; // array holds
 //hash table
 int arraySize;
```

37



## Hash taula irekiak: Eraikitzaitzailea eta hash funtzia

```
public HashTableIrekia(int size) // eraikitzaitzailea
{
 arraySize = size;
 hashArray = new LinkedList<K,V>[arraySize];
 for (int j = 0; j < arraySize; j++)
 hashArray[j] = new LinkedList<DataItem<K,V>>();
}
```

38

```
public int hashFunc(K pKey)
{
 return (k.hashCode() % arraysize); // hash function
}
```



## Hash taula irekiak: put metodoa.

```
public V put (K pKey, V pData) { // insert a DataItem
 if(! containsKey(pKey)) {
 int hashVal = hashFunc(pKey); // hash the key
 hashArray[hashVal].addFirst(new DataItem(pKey, pData));
 return null;
 }
 else {
 int hashVal = hashFunc(pKey);
 Iterator<DataItem>K, V>> it = hashArray[hashVal].iterator();
 boolean enc = 0;
 int index = 0;
 while (! (enc == 1)){ // ziur elementua badagoela!
 DataItem<K, V> elem = it.next();
 if elem.key.equals(pKey)) enc = 1;
 else index++;
 }
 V value;
 value = elem.data;
 hashArray[hashVal].remove(index);
 hashArray[hashVal].addFirst(new DataItem(pKey, pData));
 return value;
 } // end put()
}
```



## Hash taula irekiak: metodoak

```
public boolean containsKey (K pKey) {} // find item with pKey
```

```
public V remove (K pKey) {} // delete item with pKey
```

```
public V get (K pKey) {} // return key associated value or null
```



## Hash taulak: Ez dira erabili behar

- Aldez aurretik elementuen kopurua oso aldakorra denean eta ezagutzen ez denean
- Elementuak ordena jakin batean atzitu behar direnean (adibidez, txikienetik handienera)

41



## Hash taulak. Laburpena (1)

- Aplikazio askotan bilaketa eraginkorrapak behar dira, gako baten bidez (NAN zenbakia, produktu baten kodea, etabar). Gako batek ez du onartzen balio errepetitivuk
- Gako-kopurua >> elementu-kopurua
- Adibidez: gakoa: string(1 .. 20) =>  $26^{20}$  gako baina gehienez  $10^5$  elementu daudela dakigu!

42



## Hash taulak. Laburpena (2)

- Gakoa ezin da erabili array baten indizetza:
  - Elementu gehiegi (gehiengak hutsik)
  - Gainera, gakoa ez bada numerikoa, lengoaiak gehienek ez dute uzten gako hori arrayaren indizetza erabiltzea. Adibidez: “jose luis”

43



## Hash taulak. Laburpena (3)

- (A) aukera:  $10^5$  elementuren arraya (indizea 1etik 100000raino)

- Arazoak:
  - Bilaketa
  - Txertaketa
  - Ezabaketa
  - Eragiketa sekuentzialak:  $O(N)$
  - Gehienez bilaketa dikotomikoa ( $O(\log N)$ )
  - Baina txertaketa eta ezabaketa egitean elementuak mugitu behar dira =>  $O(N)$

44



## Hash taulak. Laburpena (4)

### (B) aukera: Hash taula (itxia)

- Bilaketa
  - Txertaketa
  - Ezabaketa
- Denbora ia konstantea ~  $O(1)$  (bibliografia: analisi zehatzza)
- Zeren arabera:
    - Hash funtzioa
    - Talkak
    - Taularen tamaina (karga-faktorea)

45



## Hash taulak. Laburpena (5)

### (C) aukera: Hash taula (irekia)

- K elementuren taula
  - Azpilista baten batezbesteko elementuak: N / k
- Bilaketa
  - Txertaketa
  - Ezabaketa
- Denbora ia konstantea ~  $O(N / k)$  (bibliografia: analisi zehatzza)

46



## Hash taulak. Laburpena (6)

(D) aukera. Zuhaitza DMA:

BZB (orekatua: AVL, ...)

– Denbora

- Bilaketa
- Txertaketa
- Ezabaketa

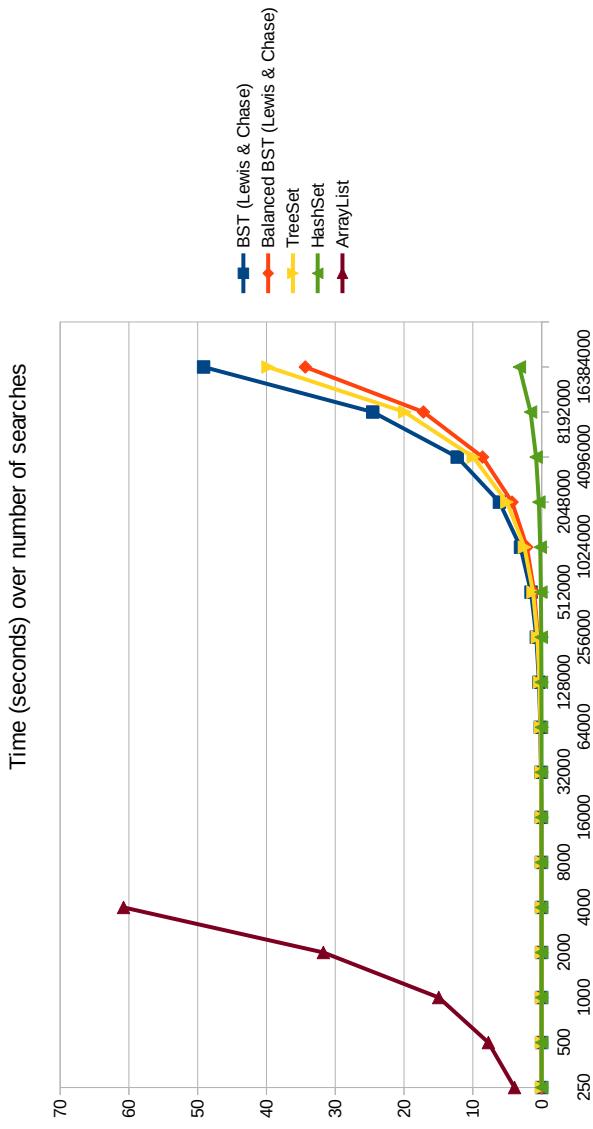
Denak:  $\sim O(\log N)$

47

### Datu-egituren eraginkortasuna (denbora segundoetan)

| Number of searches | BST (Lewis & Chase) | Balanced BST (Lewis & Chase) | TreeSet | HashSet | ArrayList |
|--------------------|---------------------|------------------------------|---------|---------|-----------|
| 250                | 0.007               | 0.004                        | 0.001   | 0       | 3.937     |
| 500                | 0.004               | 0.006                        | 0.002   | 0.001   | 7.737     |
| 1000               | 0.008               | 0.006                        | 0.004   | 0.001   | 14.95     |
| 2000               | 0.008               | 0.007                        | 0.006   | 0.002   | 31.724    |
| 4000               | 0.016               | 0.015                        | 0.012   | 0.005   | 60.775    |
| 8000               | 0.03                | 0.025                        | 0.023   | 0.009   |           |
| 16000              | 0.049               | 0.034                        | 0.041   | 0.009   |           |
| 32000              | 0.099               | 0.068                        | 0.079   | 0.01    |           |
| 64000              | 0.194               | 0.136                        | 0.158   | 0.013   |           |
| 128000             | 0.386               | 0.27                         | 0.312   | 0.026   |           |
| 256000             | 0.77                | 0.538                        | 0.623   | 0.052   |           |
| 512000             | 1.538               | 1.075                        | 1.246   | 0.102   |           |
| 1024000            | 3.072               | 2.147                        | 2.501   | 0.202   |           |
| 2048000            | 6.14                | 4.293                        | 5.043   | 0.404   |           |
| 4096000            | 12.27               | 8.594                        | 9.938   | 0.809   |           |
| 8192000            | 24.543              | 17.175                       | 19.888  | 1.612   |           |
| 16384000           | 49.137              | 34.351                       | 39.771  | 3.229   |           |

# Datu-egituren eraginkortasuna (denbora segundoetan)



## MAP Hierarchy in Java

<https://www.geeksforgeeks.org/map-interface-java-examples/>

## HashMap eta TreeMap

- Biak dira Map interfaizearen implementazioak
- Java *Collections Framework*-aren parte dira (java.util)
- Key-value bikotea darabilte datuak biltegiratzeko
- <https://www.baeldung.com/java-treemap-vs-hashtable>

06/09/2021

Datu-Egiturak eta Algoritmoak

51

## HashMap eta TreeMap. Desberdintasunak

### HashMap

- Datuak gordetzeko: arraya
- Gakoak ordenatuta egotea ez da bermatzen
- Konplexutasun-ordena:  
 $\sim O(1)$

### TreeMap

- Datuak gordetzeko: orekatutako BZBa (*Red-Black tree*)
- Gakoak ordenatuta daude beren ordena naturalean
- Konplexutasun-ordena:  
 $O(\log(n))$

06/09/2021

Datu-Egiturak eta Algoritmoak

52

## HashMap eta LinkedHashMap. Desberdintasunak

### HashMap

- Esan dugunez,  
gakoak ordenatuta  
egotea ez da  
bermatzen

### LinkedHashMap

- Estekadura bikoitzeko zerrenda  
erabilitzen du itemak gordetzeko
- Horri esker, gakoen txertatze-  
ordenari eusten dio
- Memoria gehixeago behar izaten  
du: hash erabilera ohikoaz gain,  
urreko eta ondoko itemen  
erreferentzia ere gordetzen du eta
- Eragiketen komplexutasun-ordena berdina  
da: ~  $O(1)$

## 2 (X puntu)

Liburutegi bateko liburuak, bazkideak eta maileguak errepresentatzeko bi implementazio ditugu:

### Implementazio 1: Zerrenda Estekatuak

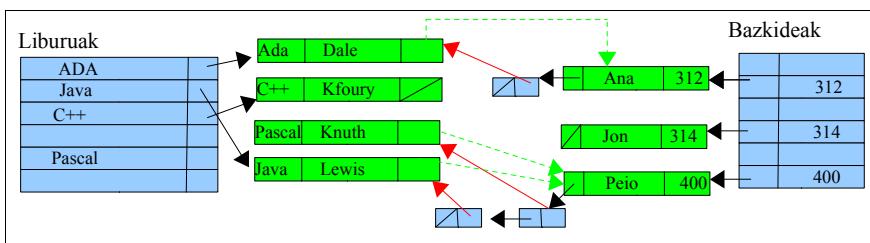
Liburutegiak bi zerrenda dauzka: liburuaren zerrenda eta bazkideen zerrenda. Bi zerrendak ordenatuta daude, lehenengo alfabetikoki liburuaren izenburuagatik, eta bigarrena bazkideen karneta zenbakiagatik.

Liburu zerrendako elementu bakoitzak izenburua, idazlea eta erakusle bat maileguan duen bazkideari dauka. (liburua, liburutegian badago erakusle hori null balioa edukiko du). Bazkide zerrendako elementu bakoitzak izena, karneta zenbakia eta bazkideen mailegu zerrenda (liburutegitik atera dituen liburuak) dauka. Mailegu zerrendako elementu bakoitzak, erakusle bat dauka maileguan daukan liburuari.

```
class Liburutegia {
 LinkedList<Bazkide> zBazkideak;
 LinkedList<Liburu> zLiburuak;
}
class Bazkide {
 String izena;
 int zenb;
 LinkedList<Liburu> zBazkideLiburuak;
}
class Liburu {
 String izenburua;
 String egilea;
 Bazkide maileguBazkide;
}
```

### Implementazio 2: Hash taulak

Liburutegi klaseak, bi hash taula dauzka: liburuak eta bazkideen taula. Hurrengo irudian, liburutegi bat aurkezten da 4 libururekin eta 3 bazkiderekin.



Liburuak taulako elementu bakoitzak izenburua, idazlea eta erakusle bat maileguan duen bazkideari dauka. (liburua, liburutegian ez badago erakusle hori null balioa edukiko du).

Bazkideak taulako elementu bakoitzak izena, karnetzen zenbakia eta bazkideen mailegu zerrenda (liburutegitik atera dituen liburuak) dauka. Mailegu zerrendako elementu bakoitzak, erakusle bat dauka maileguan daukan liburuari.

```
class Liburutegia {
 Hashtable<Integer, Bazkide> bazkideak;
 Hashtable<String, Liburu> liburuak;
}
class Bazkide {
 String izena;
 int zenb;
 LinkedList<Liburu> zBazkideLiburuak;
}
class Liburu {
 String izenburua;
 String egilea;
 Bazkide maileguBazkide;
}
```

Honakoa eskatzen da:

A) **Diseinatu eta implementatu** bazkideaTransferitu metoda liburutegiaren bi implementazioetarako.

**public void** bazkideaTransferitu (**int** zahark, **int** berk, **String** beriz);  
-- Aurre: zahark eta berk bazkide zaharraren eta berriaren karnetzen denak dira, hurrenez hurren. beriz bazkide berriaren izena da. zahark liburutegian dago, eta berk ez dago liburutegian.

-- Post: Liburutegian zahark karnetadun bazkidea ezabatu da, eta berk karnetadun bazkidea sartu da. Ezabatutako zahark bazkideak zeuzkan mailegu guztia berik bazkide berriari pasa zaizkio.

B) Esan, **modu arrazoituan**, zein den A ataleko implementazio bakoitzaren **konplexutasun-ordena**. Kontuan hartu lista sekuentzialeko metodoak ordena konstantekoak direla, eta bazkide batek eduki ditzaileen maileguen kopuru maximoa balore konstante batek mugatzen duela.

2 (4 puntos)

Tenemos 2 implementaciones para representar los libros, socios y préstamos de una biblioteca:

#### Implementación 1: listas enlazadas

La biblioteca tiene 2 listas: una para libros y otra para socios. Las 2 listas están ordenadas, la primera alfabéticamente por título del libro, y la segunda por el número de carné del socio.

Cada libro tiene su título, autor y una referencia al socio que ha tomado prestado el libro (con valor null si el libro no ha sido prestado). Cada socio tendrá su nombre, número de carné y una lista de los préstamos de ese socio. Cada elemento de la lista de préstamos es una referencia al libro prestado.

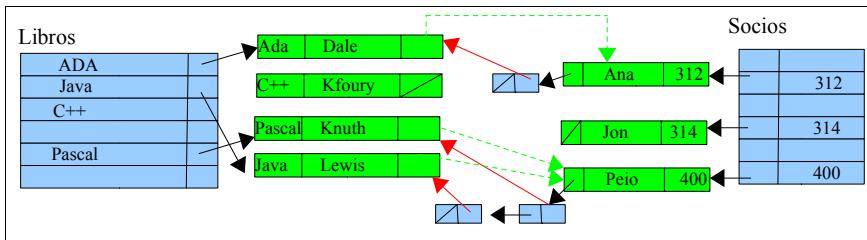
```
class Biblioteca {
 LinkedList<Socio> listaSocios;
 LinkedList<Liburu> listaLibros;
}

class Socio
{
 String nombre;
 int numero;
 LinkedList<Libro> listaLibrosPrestados;
}

class Libro {
 String titulo;
 String autor;
 Socio quienLoTiene;
}
```

#### Implementación 2: tablas Hash

La clase Biblioteca tiene dos tablas hash: una de libros y otra de socios. La siguiente figura muestra una biblioteca con 4 libros y 3 socios:



```
class Biblioteca {
 Hashtable<Integer, Socio> socios;
 Hashtable<String, Libro> libros;
}

class Socio {
 String nombre;
 int numero;
 LinkedList<Libro> listaLibrosPrestados;
}

class Libro {
 String titulo;
 String autor;
 Socio quienLoTiene;
}
```

Se pide lo siguiente:

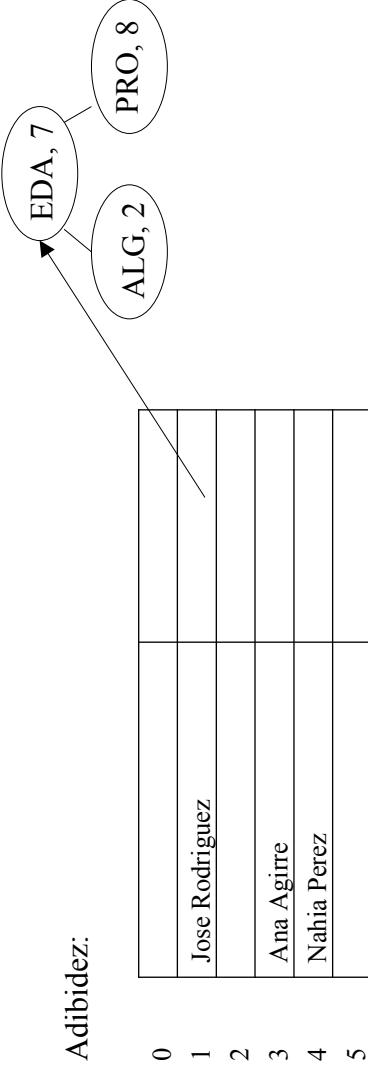
A) Diseñar e implementar el método transferirSocio en las 2 implementaciones.

```
public void transferirSocio (int carneViejo, int carneNuevo, String nombreNuevo);
// Pre: carneViejo y carneNuevo son los números de carné del socio viejo y nuevo
// nombreNuevo es el nombre del nuevo socio. carneViejo se encuentra en la
// biblioteca, y carneNuevo no.
// Post: se ha borrado el socio de número carneViejo de la biblioteca, y se ha añadido el socio de
// nombre carneNuevo. Además, se han pasado todos los préstamos de carneViejo al
// nuevo socio.
```

B) Decir, de manera razonada, cuál es el coste de cada una de las implementaciones del apartado A. Tener en cuenta que las operaciones de la clase LinkedList son de tiempo constante, y que el número máximo de préstamos que puede tener un socio viene dado por una constante.

#### 4. Lortu nota hobereneko ikasleak (1,5 puntu)

Hash taula bat dugu ikasleek lortutako notak gordetzeko. Gakoa ikaslearen izen-abizenak dira, eta irakasgaien noten informazioa bilaketa-zuhaitz bitar batean gordetzen da, irakasgaiko notaren arabera ordenatuta.



```

public class BinaryTreeNode<T> {
 protected T content;
 protected BinaryTreeNode<T> left;
 protected BinaryTreeNode<T> right;
}

public class BinarySearchTree<T> {
 protected int count;
 protected BinaryTreeNode<T> root;
}

public class Irakasgaia {
 int nota;
 String izena;
}

```

public class Irakasgaienzuhaitza extends BinarySearchTree<Irakasgaia> {  
 // Zuhaitza ordenatuta irakasgaiko notaren arabera
}

public class Ikasleak {  
 HashMap<String, Irakasgaienzuhaitza> lista;
}

public LinkedList<String> Lortu\_Nota\_Hobereneko\_Ikasleak(LinkedList<String> l)  
 // Post: emaitza 1 zerrrendako nota hoberena duten ikasleen izenak dira
}

Ondokoak eskatzen da:

- Lortu\_Nota\_Hobereneko\_Ikasleak metodoa implementatu, ikasleen zerrenda bat emanda, nota hoberena lortu duten ikasleen zerrenda bueltatuko duena (edozein irakasgaitan). Adibidez, demagun 8 dela nota maximoa irakasgai guztietan, eta kasu horretan 8 atera dutenen zerrenda aterako litzateke.
- Kostua kalkulatu, modu arrazoiutan.

## Notak

```

public class BinaryTreeNode<T> {
 protected T content;
 protected BinaryTreeNode<T> left;
 protected BinaryTreeNode<T> right;
}

public class BinarySearchTree<T> {
 protected int count;
 protected BinaryTreeNode<T> root;
}

public class Irakasgaia {
 int nota;
 String izena;
}

public class Irakasgaienzuhaitza extends BinarySearchTree<Irakasgaia> {
 // Zuhaitza ordenatuta irakasgaiko notaren arabera
 public Integer notaHoberena() { // Post: emaitza zuhaitzeko nota hoberena da
 return notaHoberena(this.root);
 }

 private Integer notaHoberena(BinaryTreeNode<Irakasgaia> nodo) {
 if (nodo.right == null) return nodo.content.nota;
 else return notaHoberena(nodo.right);
 }
}

public class Ikasleak {

 HashMap<String, Irakasgaienzuhaitza> lista;

 public LinkedList<String> Lortu_Nota_Hobereneko_Ikasleak (LinkedList<String> l) {
 // Post: emaitza 1 zerrendako nota hoberena duten ikasleen izenak dira
 // zuhaitzean

 LinkedList<String> emaitza;
 Iterator<String> it = l.iterator();
 Integer notaMax = -1;

 while (it.hasNext()) {
 String ikaslearenIzena = it.next();
 Irakasgaienzuhaitza a = lista.get(ikaslearenIzena); // hash-taulan bilatu
 Integer nota = a. notaHoberena();
 if (nota > notaMax) {
 // zerrenda berri batekin hasiko gara
 emaitza = new LinkedList<String>();
 emaitza.add(ikaslearenIzena);
 }
 else if (nota == notaMax)
 emaitza.add(ikaslearenIzena);
 // nota txikiagoa baldin bada, ez egin ezer
 } // while
 return emaitza;
 }
}

Kostua:
N: zerrendako elementu-kopurua
M: ikasle baten batezbesteko irakasgai-kopurua

O(N.log2M)
Kostu arrazoitua: zerrendako elementuak banan-banan aztertuko dira (N denbora). Irakasle bakoitzaz hash-taulan bilatu da (O(1)), eta bakoitzeko bere nota hoberena bilatu da (kostu logaritmikoa).

```

#### 4. Ebaluatzalea (1,5 puntu)

Era honetako aginduen sekuentzia dugu: “x = y + z”. Hash-taula batean aldagaien balioak daude. Programa bat egin nahi dugu, aginduak ebaluatu eta hash-taula egunerazteko.

```
public class Agindua {
 String ize1;
 String ize2;
 String ize3;
 String eragile; // balio hauetatik bat: "+" , "-" , "/" eta "*"
 // era honetako agindua adierazten du:
 // ize1 = nombre2 ize ize3
}

public class AldagaienHashTaula extends HashMap<String, Integer> {
 public void ebaluatu (SimpleLinkedList<Agindua>)
}
```

Adibidez, aginduen zerrenda hau emanda:

Aginduen zerrenda: (x = x + j; contNum = contNum + x; y = contNum + x)

Hash-taula era honetan aldatuko litzateke:

|   |         |   |   |         |    |
|---|---------|---|---|---------|----|
| 0 | j       | 5 | 0 | j       | 5  |
| 1 | x       | 3 | 1 | x       | 8  |
| 2 |         |   | 2 |         |    |
| 3 | contNum | 0 | 3 | contNum | 8  |
| 4 | y       | 7 | 4 | y       | 16 |
| 5 |         |   | 5 |         |    |

↑

Ondokoak eskatzen da:

- Ebaluatu metodoa implementatu.
- Bere kostua kalkulatu **era arrazoituan**.