ESTRUCTURA DATOS

estructura datos

FORMAL SPECIFICATIONS OF AN ADT

Data logic = Cierto ____ | Falso ____ + When the parameter is a generator you must commit the output

or, and :: Lógico → Lógico → Lógico (Role very important in Haskell because of new commands)

not :: Lógico → Lógico (Bool)

not Cierto = Falso // not Falso = Cierto

andor x y = z

orand

false = not cierto.

UNIT 1. INTRODUCTION

1.1. Terminology:

- A type (sort) is a set of values → boolean < true / false
  - int by values 0, -1, 1, -2, 2 ... ; int[] composed by {1,2}, {1,2,3}, {1,4,6}
  - Par (int, char) composed by (1, 'a'), (0, 'g')
- A datum is a value belonging to concrete type (true, 0, {1,2})
- An Abstract Data Type (ADT) is a type together with a set of operations for managing its data, without taking into account any implementation detail. Each operation is defined in terms of its inputs and outputs: Boolean ADT = {True, False, and, or, not} ; Naturals ADT = {0, Suc, +, -, <, ...}
- A Data Structure is a physical implementation of one or more ADT's. Each operations defined into an ADT is associated to one or more functions/procedures/methods written with a concrete programming language. It's data is usually stored in central memory. When it is stored into secondary memory we have File

1.2. Data Structures and Efficiency

- A computer program can be decomposed as: Program = Algorithm + Data Structures [N. Wirth]
  Algorithm = Logic + Control
- The most important task of a program (VERY IMPORTANT) consists in storing and retrieving information. It is crucial to organise data in way that information can be manipulated as more efficient as possible.
- Algorithms describe the sequence of actions to be applied on data and they have 2 components: definition of operations and implementation.
- A good choice of appropriate data structures to manipulate the data involved in a program, as well as a clever design might imply that such program is executed in a few seconds offering the pursued results.
- Some criteria which will help us to select a good Data Structure for a given problem:
  - To analyse the problem to determine the Data Structure
  - To decide the basic sets of operations to be applied on each Data Structure
  - Inserting, deleting, searching... data, are considered basic operations on a Data Structure
  - To select the Data Structure which minimizes the computational cost

1.3 Data Structures and Object Oriented

- O.O. Programming becomes a suitable paradigm for implementing a Data Structure thanks to its wide repertoire of resources and tools for performing this task:
  - The notion of class allows us to clearly define and create a Data Structure
  - Information hiding enables the possibility of hiding most implementation details of the Data Structures to the final user.
  - Inheritance is useful for creating new Data Structure from previous existing ones, as well as for building generic D.S.
  - Polymorphism allows to change the behaviour of a given operation depending on the D.S where it is applied

UNIT 2: FORMAL DESCRIPTION OF DATA TYPES

2.1 Introduction:

- The definition of an ADT is performed independently of its implementation. We can use formal languages which sometimes offer an implementation of the ADT
- There exist a mathematical definition for int used for implementing this concept with a programming language. It is mandatory to know such mathematical description for using in JAVA. Something similar occurs with other types like boolean, real, array, arraylist... The more complex an ADT is, the most required is too its formal definition.
- It's important separate the specification from its implementation(s)

2.2 Formal Specification of an ADT.

- GOAL: To establish what a Data Type is by means of an algebraic model.
  - We define the meaning or semantics of the operations by using a set of equations or axioms ($\forall n, m \in Nat$) add(Suc(m), n) = Suc(add(m,n))
  - In order to define the meaning of any ADT, the set of axioms must be sufficient (enough), any property of the ADT should be proven from the set of axioms, but it will not be possible to demonstrate false properties
  - Axioms describe in an abstract way a minimal set of constraints or conditions which must be verified by any concrete implementation of the ADT

ADVANTAGES

- Users share the same (unique) interpretation of the ADT
- Programs making use of the ADT admit formal (automatic) verification.
- Possibility of using tools for executing specifications → prototype generation.

ELEMENTS APPEARING IN A FORMAL SPECIFICATION OF AN ADT

- 1. Set of types required in the definition → Interested Type or Intended Type (IT) → Sorts or Domains.
- 2. Operations for manipulating the data of the IT → Provide names and prototypes or shapes.

We can distinguish the following kinds of operations:

- Generators or constructors: Used to built all data of the IT. Data are really the terms produced with these operations and they can't no seen as applications which produce an output
- Modifiers: Useful for obtaining data of the IT from other operations which aren't generators.
- Consult operations: Return objects different from the IT
  - Predicates: They provide a Bool value
  - Selectors: They provide attributes of data collected in a IT.
- 3. Sufficient set of axioms defining operations and their properties. It is usual that some axioms be recursive.
  - Axioms are mandatory for proving that any free term is always equivalent to other one built only with generators → inform about equivalence

FREE AND NOT FREE GENERATORS

- Generators of an ADT are free if different terms always represent different data.
  - Not free generators force us to establish a set of axioms for describing the equality among basic terms, called equivalence axioms required too for identifying non valid terms. They introduce an equivalence relation among terms, where there might exist a particular class collecting the set of all possible non valid
  - Equivalence → shape Suc(Suc(... Suc(zero))) or Pre(Pre(... Pre(zero))) called canonic representative → terms in normal form

HASKELL INSERT:

```
insert :: List x → List y → List a
insert   x        y  =  x : y : List.
```

30th October 2017

1.) times (specified) nf_Inte

```
data Nat = zero | Suc Nat

times :: Nat → Nat → Nat

fact :: Nat → Nat
fact:      Zero → SucZero.
fact: Suc( x) → times (fact x) ( Suc x)
```

```
data Inte = Z | S Inte | P Inte

nf_Inte :: Inte → Inte.

absI, aux_absI :: Inte → Nat
  absI  x  = aux_absI (nf_Inte x)
aux_absI :: Inte → Nat.
aux. absI   Z   → Zero.
aux_ absI  S(x) → Suc (aux_ abs x)
aux_ absI  P(x) → Suc (aux_ abs x)
```

```
factabsI :: Inte → Inte.
factabsI  Z   = Zero
factabsI  X   = aux_ fact (fact( abs_ I x))
aux_ fact :: Nat → Inte
aux_ fact  Zero  = Z
aux_ fact  Suc(x) = S( aux_ fact x)
```

EXERCISES

andor: (x ∧ y) ∨ z

andor (true, true, c) = true

andor (true, false, c) = c

andor (false, false, c) = c

---

andor (a, b, c) = c

orand (false, a, b) = false

orand (false, b, true) = b

orand (true, b, true) = true.

EXAMPLE IN HASKELL: NATURAL NUMBERS:

data Nat = Cero | Suc Nat deriving Show

suma :: Nat → Nat → Nat. Only 2 formats.

suma Suc(Suc Cero)) (Suc Cero) ⟶ Suc (Suc (Suc Cero))

↳ ecuation suma Cero x = x
suma (Suc y) x = Suc (suma x y)

Recursion parameter

Example exercise:

inf Cero = Cero

inf (Suc Cero) = Suc Cero

inf (Suc (Suc Cero)) = Suc (Suc Cero)

inf (Suc (Suc (Suc x))) = inf x

geed Cero = Suc (Suc Cero)

geed Suc (Suc Cero) = Suc Cero

geed Suc Cero = Cero

geed Suc (Suc (Suc x)) = geed x

geed Cero = Suc Cero

geed Suc Cero = Suc (Suc Cero)

geed Suc (Suc Cero) = Cero

geed (Suc (Suc (Suc x))) = geed x

TEST IF THE INT IS IN NORMAL FORM

test :: Inte ⟶ Bool

test z = True

test (P(Sx)) = False

test (S(Px)) = False

test Px = test x

test Sx = test x

FACTORIAL

ParityFactorial :: Nat → Nat

ParityFactorial Cero = Suc Cero

ParityFactorial Suc Cero = Suc Cero

ParityFactorial (Suc (Suc x)) = times (parifactorial (x)) (Suc (Suc x))

ParityFactorial :: Nat → Nat

ParityFactorial Cero = Cero

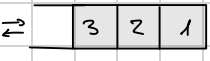
ParityFactorial Suc Cero = Suc Cero

ParityFactorial (Suc (Suc x)) = times (parifactorial (x)) (Suc (Suc x))

We add here ParityFactorial Suc (Suc Cero) = Suc (Suc Cero)

JAVA LISTS

generators { `[]` ↔ empty list ; `(x : t)` ↔ (x = element, t = always a list) }



LIFO → last input is first output.

`head :: [a] → a`

`head [] = error "...."`

`head (x : ___) = x`

`tail :: [a] →`

`(3 : (2 : (1 : [])))`

`++ :: [a] → [a] → [a]`

`list ++ [] = list`

`list ++ (x : t) = (x : list) ++ t` ← WHAT IS IT DOING?

↳ `l ++ (x : t) = (l ++ [x]) ++ t`

LOW LEVEL < Dynamic / Static

SUMMARY

IMPORTANT → LIFO (last input first output)

`list | [] | [] | (x : t) | head | tail` Ex: tail output is the list containing 1.

`Stack | Stack a | EStSV | (x :/ s) | top | pop`

`Queue | Queue a | EQuCV | (q :. x)`

`qsq :: Queue a → Stack a → Queue a`

`qss :: Queue a → Stack a → Stack a` } recursscon needed.

`sqq :: Stack a → Queue a → Queue a`

`sqs :: Stack a → Queue a → Stack a`.

`cq :: Queue a → Queue a → Queue a`

`cq q EQn = q`

`cq q1 q2 = cq (q1 :. (first q2)) (rest q2)`

`cq q1 (q2 :. x) = cq (q1 :. x) q2`

↳ `cq q1 q2 :. x = cq (tQn :. 1) (tQn :. 2 :. 3) :. 4 = cq (tQn :. 1) (tQn :. 2) :. 3 :. 4 = cq (tQn :. 1) (tQn) :. 2 :. 3 :. 4 =`

`:. EQn :. 1 :. 2 :. 3 :. 4`

`sq :: Stack a → Queue a`

`sq ESt = EQn`

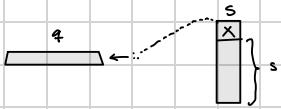
`sq (x :/ s) = (sq s) :. x`

`sq2 s = aux s EQn` → `aux :: Stack a → Queue a → Queue a`

`aux (x :/ s) q = aux s (q :. x)`

`aux EST q = q`

qsq → Queue a → Stack a → Queue a



STUDY 1ST PROGRESS TEST

NOVEMBER 8TH, 2022

1) SPECIFICATION:

a) (NON-FREE generators) data IntI = Z | S IntI | P IntI deriving subI:: IntI → IntI → IntI Input = Output.

subI: IntI → IntI → IntI

sub I x Z = x

Sub I (Sx) (Sy) = SubI x y

sub I x (Sy) = P (subI x y)

subI Px Py = subI x y

sub x Py = S (subI x y)

b) Haskell.

op 22 (EQu: 1:2:3) (4:/ES) → 3:/ (op22 EQu: 1:2 4:/ES) = 3:/(2:/(1:/ ES)) ¿Y el 4?

op 22 EQu 4:/5:/6:/ES → 6:/ (op22 EQu 4:/5:/ES) = 6:/(5:/(op22 EQu 4:/ES)= 6:/(5:/(4:/ES) = 5:/(6:/ES)

op22 (EQu: 1:2:3) (4:/5:/6:/ES) = 3:/2:/1:/5:/6:/ES

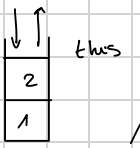
2) a) HIGH LEVEL: public static <E> Pila<E> op22 (Cola<E> q, Pila<E> s)

{ Pila<E> s1; Cola<E> q1; ... }

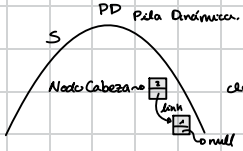
b) LOW LEVEL: public void op22 (CD<E> q)

{ Nodo<E> aux = null; .... }

JAVA
- LOW
  - STATIC → array.
  - DYNAMIC → links, pointers
- HIGH



this



class Nodo <E>
{ Info E; Nodo <E> Siguiente.

public class Nodo (x, p)
{ Info = x; Siguiente = p; }

public PD ()
{ NodoCabeza = Null }

public PD (E x, PD<E> s)
{ if (e != null)
New Nodo (x, s. NodoCabeza)
{ PD <E> pd = new PD<E> ();

this. NodoCabeza = new Nodo (x, s. Nodo Cabeza)

Haskell []

Preserve the state of the parameter → Do a clone()

Public void Apila(E e); Public void APila (E x, Pila <E> p) ; public Pila <E> APila (E e) ; Public Pila <E> APila (E e, Pila <E> p)

(Codigo en la practica)

{ if (P. EsVacia())
p. NodoCabeza = x. NodoCabeza
else {
s = new PD (p. tope(), p. DesaPila)
s. NodoCabeza = new Nodo (x, s. NodoCabeza)
{ return s; }

Ejercicio:

qsq :: Queue a -> Stack a -> Queue a

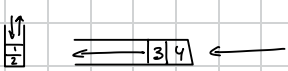
qsq q Est = q

qsq q x:/s = (qsq q s) :. x ← NOT GOOD BECAUSE STACK INFO IS INSERTED IN REVERSE WAY.
↳ qsq (q:.x) s

sqs :: Stack a → Queue a → Stack a

sqs Est q = qs q

(x :/ s) q = x :/ (sqs s q) Transforming queue into a stack.



sqs (1 :/ 2 :/ Est) (EQu :. 3 :. 4)
= 1 :/ (sqs 2 :/ Est (EQu :. 3 :. 4)
= 1 :/ 2 :/ (sqs Est (EQ :. 3 :. 4)

qs :: Queue a → Stack a

qs EQu = Est

qs q :. x = x :/ (qs q) → ejemplo qs EQu :. 3 :. 4 → 4 :/ (qs EQu :. 3) = 4 :/ 3 :/ (qs EQu) = 4 :/ 3 :/ Est.

Use this line:

infixl 5 :. lo usamos como encola (por comodidad)

ordenar :: (Ord a) => Cola a -> Cola a

ordenar CV = CV

ordenar q = (ordenar (quitar q (menor q))) :. (mayor q)

If we can compare objects its classes are comparable.

infixr 5 :/ insertar.

polish notation:

2 * 3 + 4 = 10 → 2 3 * 4 + = 10

2 * (3 + 4) = 14 → 2 (3 4 +) *

IMPORTANT THINGS:

* Generators of an ADT are free if different terms always represent different data

EXERCISES: REVIEW UNIT 2

① Bool3 ADT

data Bool3 = True3 | False3 | Maybe3 deriving (Show)

Example for using: Function to combine 2 values value1 = True value2 = False value3 = Maybe

andBool3 :: Bool3 -> Bool3 -> Bool3.

andBool3 True3 True3 = True3

andBool3 False3 x = False3

andBool3 x False3 = False3

andBool3 x y = Maybe3

andBool3 value1 value2 = False

andBool3 value1 value3 = Maybe.

② Bool10 ADT → Not exist false, True is value <10 but 0 = False.

Bool10 :: Bool10 -> Bool10 -> Bool10.

Bool10 (True10 x) (True10 y) = True10 (min x y)

Bool10 _ _ = False10

(3) Nat ADT: greater, equal, even, add, times, pow, fac, sub, divide, mcd, mcm

```
greater :: Nat -> Nat -> Nat.
greater  Zero  x  =  x
greater  x  Zero  =  x
greater  (Suc x)  (Suc y) = Suc (greater x y)

equal :: Nat -> Nat -> Bool

times :: Nat -> Nat -> Nat
times  Zero  x  =  Zero
times  (Suc y)  x  =  add  x  (times  y  x)

add  Nat -> Nat -> Nat
add  x  Zero  =  x
add  x  (Suc y)  =  Suc (add  x  y)

pow :: Nat -> Nat -> Nat
pow  x  Cero =  Suc Cero
pow  x  (Suc y) =  times  (pow  x  y)  x

factt :: Nat -> Nat
factt  Cero =  Suc Cero
factt  (Suc x)  =  times  Suc x  (factt x)
```

HACER DIAPOSITIVA 3.1. QUEUES

REPASO:

Examen 8th November 2022.

② Specification:

a) SubI :: Inte -> Inte -> Inte

```
subI  x  z = Px
subI  x  Sy =  P (subI Px  y)
subI  x  Py =  S (subI Sx  y)
subI  Px  Py =  subI (x  y)
subI  Sx  Sy =  subI (x  y)
```

b)
```
infixr 5:/ data Stack a = ES | a:/ (Stack a) deriving Show
infixl 5:/ data Queue a = EQu | Queue :.a deriving Show.
op22 :: Queue a -> Stack a -> Stack a
op22 (q:.x)  s  =  x:/ (op22  q  s)
op22  q  (x:/y:/s) = y:/(op22 q  x:/s)
op22  __  __  = ES
```

b) Valuate the following expressions

1) op22 (EQ :. 1 :. 2 :. 3) (4:/ES) = 3:/(op22 (EQu :. 1 :. 2 (4:/ES)) = 3:/2:/(op22 EQu :. 1 4:/ES)
3:/2:/1:/(op22 EQ 4:/ES) = 3:/2:/1:/ES

2) op22 EQu (4:/5:/6:/ES) = 5:/6:/ES

3) op22 (EQu :. 1 :. 2 :. 3) (4:/5:/6:/ES) = 3:/2:/1:/(op22 EQ 4:/5:/6:/ES) = 3 2 1 5 6 ES

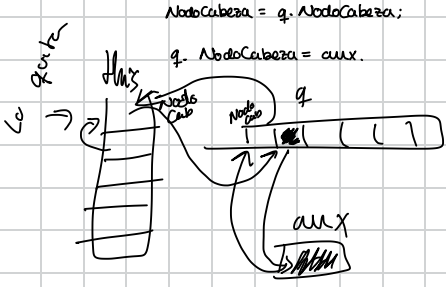
2. IMPLEMENTATION:

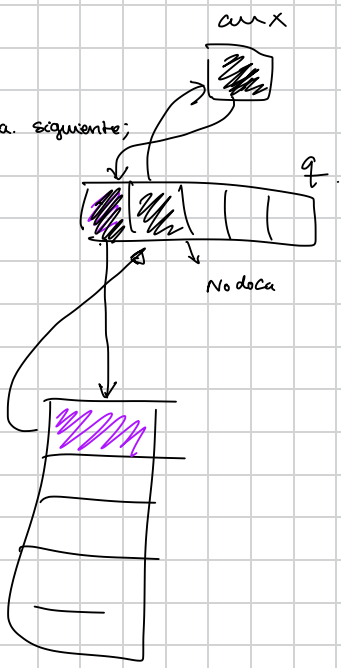
a) 2 efficient java methods:
1) No more variables than the ones declared in the templates
2) Try {...} catch (TADVacioException ex) {ex.printStackTrace();} must be used when necessary.
3) Input in HIGH level method will remain unchanged after calling such method.
4) LOW level methods will be able to alter the content of the input, object thus encode the PD class

a) High level:

```
public static <E> Pila<E> op22 (Cola <E> q, Pila <E> s)
  { Pila <E> s1 = (Pila <E>) s.clone();
    Cola <E> q1 = (Cola <E>) q.clone();
  try { if( ! s1.EsVacia()) s1 = s1.Despila();
        while ( ! q1.EsVacia())
            s1.Apila(q1.Cabeza());
            q1 = q1.resto();   Me lo tengo que saber.
        }
    { catch (TADVacioException ex) } ex.PrintStackTrace(); {
    return s1; }
```

b)
```
public void op22 (CD <e> q)
  { Nodo<E> aux = null
    if ( NodoCabeza != null) NodoCabeza = NodoCabeza.siguiente;
      while (q.NodoCabeza != null) {
        aux = q.NodoCabeza.siguiente;
        q.NodoCabeza.Siguiente = NodoCabeza;
        NodoCabeza = q.NodoCabeza;
        q.NodoCabeza = aux.
```





```
Public void Desapila () throws TADVacioException
    { if (NodoCabeza == null) throw new TADVacioException();
      else { NodoCabeza = NodoCabeza.Siguiente;
             }
    }
```

op22 :: Queue a → Stack a → Stack a

op22 (q:.x) ES = x:/ES

op22 (q:.x) (y:/s) = x:/(op22 q (y:/s)

op22 EQu (x:/y:/s) = y:/(op22 EQu (x:/s)

op22 ___ ___ = ES

Ex: op22 (EQu:.1:.2:.3) (4:/5:/6:/ES) = 3:/(Op22 (EQu:.1:.2) (4:/5:/6:/ES)) = 3:/2:/1:/(Op22 EQu (4:/5:/6:/ES)

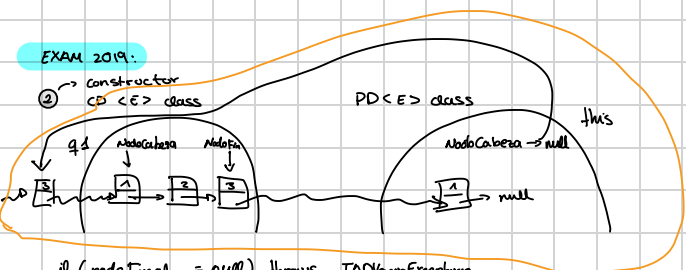
(3:/2:/1:/5:/6:/ES) (Exam solution)

Ex: op22 (EQu:.1:.2:.3) (4:/5:/6:/ES) = 3:/2:/1:/5:/6:/ES

→ Lucian

```
op22 :: Queue a → Stack a → Stack a
op22 EQu EST = EST
op22 EQu (x:/s) = s
op22 (q:.x) s = x:/op22 (q s)
```

EXAM 2019:



this (after calling constructor)

```
if (nodoFinal = null) throws TADVacioExeption
else ( do { nodoFinal.Siguiente = new Nodo<E> ( x, null)
            nodoFinal = nodoFinal.Siguiente.
```

EXAM NOVEMBER 2020

2) quevexam :: Queue Int → Queue Int

quevexam (q:.x:.y) = (q:.y:.x)

quevexam ___ = quevexam (EQu:.1:.2)

quevexam (EQu:.3) = quevexam (EQu:.1:.2) = (EQu:.2:.1)

quevexam (EQu:.3:.2:.1) = (EQu:.3:.1:.2)

OTHER VERSION:

quevexam :: Queue Int → Queue Int

quevexam EQu
quevexam EQu:.x } EQu:.2:.1

quevexam (Equ:.x:.y) = Equ:.y:.x

quevexam q = (first q) :. quevexam (rest q)

Necesito más operaciones auxiliares

b) HIGH CEVEL:

```
public static Cola<Integer> queueexam (Cola<Integer> q)
{ Cola_Dinamica<Integer> sol = new Cola_Dinamica<Integer>();
  try {
    if( q.EsVacia || q.Resto().EsVacia() )
     { sol.EnCola(2);
       sol.EnCola(1); }
     else { while (!q.Resto().Resto().EsVacia())
          { sol.EnCola(q.Cabeza());
            q = q.Resto();
            Integer x;
            x = q.Cabeza();
            q = q.Resto();
            sol.EnCola(q.Cabeza());
            sol.EnCola(x); }
  } catch (TADVaccioException er) { System.out.println("TAD Lineal vacio"); }
    return sol;
  }
```

DATA STRUCTURE

1ST PROGRESS TEST October 30th, 2019

① Haskell:

```
data N1 = One | Next N1

data InteN1 = None | Pos N1 | Neg N1.

leqN1 :: N1 -> N1 -> Bool
leqN1 One x = True
leq N1 (Next x) One = False
leq N1 (Next x) (Next y) = leq N1 x y
```

b) leq InteN1 :: InteN1 -> InteN1 -> Bool.

```
leqInte N1 None Pos x = True
leq Inte N1 None Neg x = False
leq Inte N1 Pos x Pos Y = leqInteN1 x y
leqInteN1 Neg x Neg Y = leqInteN1 x y
leqInte N1 Pos x Neg Y = False
leq Inte N1 Neg x Pos x = True.
```

c) add N1 :: N1 -> N1 -> N1

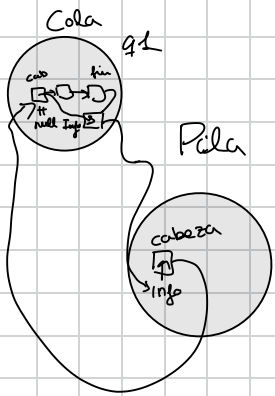
```
addN1 One x = Next x
addN1 Next x y = Next(addN1 x, y)
```

d) addmcx :: InteN1 -> N1 -> InteN1.

```
addmcx None x = Pos x
add mcx (Pos x) y = Pos (addmcx x y)
addmcx (Neg One) One = None
addmcx (Neg One) Next x = Pos x
addmcx (Neg (Next x)) One = Neg x
addmcx (Neg (Next x)) (Next y) = addmcx (Neg x) y
```

② JAVA

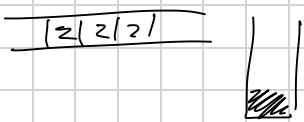
```
publicc   Pila_ Dinamica (Cola_ Dinamica <E> q)
   { Cola_ Dinámica <E>   q1 = (Cola_ Dinámica <E>) (q. clone());
      NodoCabeza = null;
      if (q1. NodoCabeza != null)
         { q1. NodoFinal. Siguiente = new  Nodo <E> (q1. NodoCabeza. Info, null);
           NodoCabeza = new  Nodo <E> (q1. Nodo Final . Info , q1. NodoCabeza): }
      }
```



a) october 19 :: Cola a → Pila a

october 19 CV = PV

obtober 19 q:.x = x:/ (o19 q:.x ((cabeza q:._):/ PV))



o19 Cola a → Pila a → Pila a

o19 CV p = p

o19 q:.x p = o19 q x:/p

cabeza :: Cola a → a

cabeza CV:.a = a

cabeza q:._ = cabeza q

october19 (CV:.1:.3:.2) = 2:/ o19 (CV:.1:.3:.2)

(Cabeza q:.):/PV

1 :/PV

2:/1:/3:/2:/1:/PV

c) publicc static <E> Pila <E> october 19 (Cola <E> q)

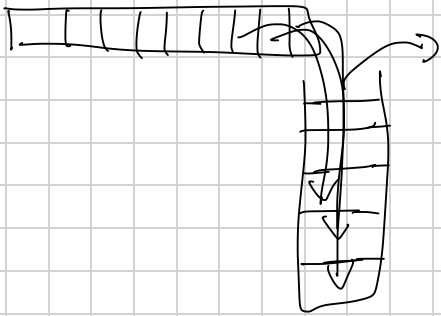
```
   { Cola <E>   q1 = (Cola <E>) (q. clone());
     Pila <E> p1 p2;   p1 = new  Pila_ Dinámica <E>( );  p2 = new Pila_ Dinamica <E>();
     if(!q1. EsVacia())
       {
        try { E first, aux;  first = q1. Cabeza(); aux = first;
              While (!q1.EsVacia())
                { aux = q1. Cabeza(); q1= q1. resto (); p1. Apila (aux); }
                p2. Apila ( first)
              while (!p1. EsVacia()) {   p2. Apila(p1. tope);  p1 = p1. Desapila(); }  P2. Apila(aux);
        catch — — — —
```

EXÁMENES

8th November, 2022.

④ a) subI :: Inte → Inte → Inte.

```
subI   x      Z     =  x
subI   S(x)   S(y)  =  subI  x  y
subI   P(x)   P(y)  =  subI  x  y
subI   x      P(y)  =  S(subI x y)
subI   x      S(y)  =  P(subI x y)
```



b) op22 (EQu:.1:.2:.3) (4:/ ES) → 3:/2:/1:/ES

op22 (EQu) (4:/5:/6:/ES) → 5:/6:/ES.

op22 (EQu:.1:.2:.3) (4:/5:/6:/ES) → 3:/2:/1:/5:/6:/ES

Queue stack stack

2

a) public static <E> Pcola<E> op22 (Cola<E> q, Pila<E> s)

```
{Pila<E> s1 = (Pcla<E>) s.clone();   // Para salvar lo que entra como parametro
 Cola<E> q1 = (Cola<E>) q.clone();
  try{
     if(!q1.EsVacia()) s1 = s1.Desapila.
        while (!q1.EsVacia())
         { int aux, first;
           s1.Apila(q1.cabeza()); q1 = q1.Cabeza();
         }
     {catch (TADVacioExcepcion ex) {ex printTrace();}
    return s1;
```

b) public void op22 (CD<E> q){

```
Nodo<E> aux = null;

if(nodoCabeza != null) NodoCabeza = NodoCabeza.Siguiente.
while (q.NodoCabeza != null){
    aux = q.NodoCabeza.Siguiente; q.NodCabeza.Siguiente = NodoCabeza
    NodoCabeza = q.NodoCabeza; q.NodoCabeza = aux;}
```

26 Octubre 2021:

1) NAND (negation of the conjunction)

```
nfBoolNAND :: BoolNAND → BoolNAND
nfBoolNAND  Yes              = Yes
nfBoolNAND  Nand Yes     Yes = Nand Yes Yes
nfBoolNAND  Nand Yes     Nand Yes Yes = Nand Yes Yes
nfBoolNAND  Nand (Nand Yes Yes)   Yes) = Yes
```

2) mix 21 :: Stack a → Queue a → Queue a

```
mix 21  (x:/s)   (q:.y) = (mix 21  (y:/s) q) :. x
mix 21  (x:/s)   q      = mix 21   s   q
mix 21    ES     q      = q
```

a) mix21 ES (EQu :. 3 :. 4 :. 5) → (EQu :. 3 :. 4 :. 5)

mix 21 (1:/2:/ES) EQu → EQu

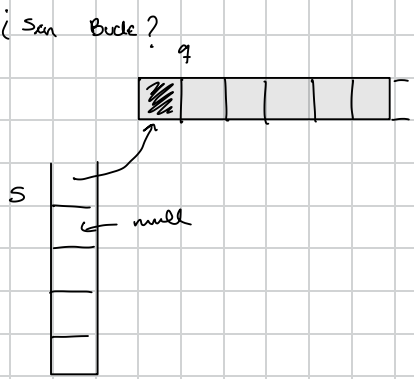
mix 21 (1:/2:/ES) (EQu :. 3) → EQu :. 1

mix 21 (1:/2:/ES) (EQu :. 3 :. 4 :. 5) → EQu :. 4 :. 5 :. 1

A) HIGH LEVEL

```
publico static <E> Cola <E> mix21 (Pila <E> s, Cola <E> q)
  { Pila <E> s2 = Pila <E> s.clone();
    Cola <E> q2 = Cola <E> q.clone();
    while / if( !s2.EsVacia() && !q2.EsVacia());
        { q2. = q2.Rastoc); q2. Encola( s. Tope()); }
        return q2;
```

¿ Sin Bucle ?



b) Low level:

```
publico void mix 21 (PD <E> s)
    if ( NodoCabeza != null && s. NodoCabeza != null) {
    Publico Nodo <E> = new Nodo< E> aux;
     aux = NodoCabeza;
     s. Nodo Cabeza. Sgquiente = null;
      Nodo Final. siguiente = s. Nodo Cabeza
      Nodo Final = Nodofinal. sgquiente
      NodoCabeza = NodoCabeza. sgquiente;
```

30th October 2017

1) times (specified) nf_ Inte

```
data Nat = zero | Suc Nat

times :: Nat -> Nat -> Nat

fact :: Nat -> Nat

fact:  zero -> SucZero.

fact: Suc( x) -> times (fact x) ( Suc x)
```

```
data Inte = Z | S Inte | P Inte

nf_ Inte :: Inte -> Inte.

absI , aux_ absI :: Inte -> Nat

  absI  x  = aux_ absI (nf_ Inte  x)

aux_ absI :: Inte -> Nat.

aux. absI   z   -> Zero.

aux_ absI   s(x) -> Suc (aux_ abs  x)

aux_ absI   P(x) -> Suc ( aux_ abs x)
```

```
factabsI :: Inte -> Inte.

factabs I   Z   = Zero

factabs I   X   = aux_ fact (fact( abs_ I  x))

aux_ fact :: Nat -> Inte

aux_ fact  Zero  = Z

aux_ fact  Suc(x)  = S ( aux_ fact x)
```

2) public static Pila < Integer> interRPN (Pila exp) (HIGH LEVEL

```
{ Pila  aux  = new  PD();
  Pila <Integer> sol = new PD<Integer>();
  Integer  x,  y , z;
try { while (! exp. EsVacia ())
  { if ( exp. Tope () instanceof  Integer)
        aux. Apila (exp. Tope());
    else { X = (Integer) aux . Tope ();
           aux = aux. Desapila ();
           y = (Integer). Tope ();
           aux =  aux. Desapila();
           z   =   ( eval (y , (char) exp Tope() , X );
           aux. Apila (z); sol. Apila (z);
           }
           exp = exp. DesaPila ();
      sol. sol. Des Apila ();
  } catch (TAD. ->) { sout (TAD...
    return sol;
```

3) NO REPEAT.

HASKELL:

```
create:: (Eq a) -> Queue a -> Quasilist a
create Eq = Eql
create q:x = Iql x (remove x (create q))

remove:: (Eq a) -> a -> Quasilist a -> Quasilist a
remove _ Eql = Eql
remove x (Iql y q1) = < if x == y then q1
                      < else Iql y (remove x q1
```

Eq :. 1 :. 2 :. 1 :. 3

Iql 3 (remove 3 (Iql 1 :. (remove 1 (Iql 2 (remove 2 (Iql 1 remove 1 Eql)

Iql 3 Iql 1 Iql 2 Eql

JAVA:

```
public class Quasilist <E> {
  public int longitud; Public Nodo <E> NodoCabeza;
  public Quasilist (ColaDinamica <E> q)
    { longitud = 0; nodoCabeza = null; Nodo <E> aux 1 = q. NodoCabeza;
      while ( aux 1 != null)
        { Nodo <E> aux 2 = aux 1. Siguiente;
          while ( aux2 != null && !aux 2. info. equals(aux 1. Info)
                  aux 2 = aux 2. siguiente;
          if (aux 2 == null) { longitud ++;
          NodoCabeza = New Nodo <E> (aux 1 . Info, NodoCabeza); }
          aux1 = aux1 . Siguiente;
        }
    }
```

5th November 2018

1. Nat ADT leqN x y = x <= y // addN // subN (Sum negatives)

IN = Pos Nat | Neg Nat

abs :: IN -> IN

abs x = x

abs Neg(x) = Pos(x)

leqIN :: IN -> IN -> Bool

leqIN pos(x) pos(y) = leqN x y

leq pos(zero) neg(zero) = True

leq pos(x) neg(y) = False

leq neg(x) pos(y) = True

leq neg(x) pos(x) = leq x y

leq :: Nat -> Nat -> Bool

leq zero zero = True

leq zero Suc(x) = True

leq Suc(x) zero = False

leq Suc(x) Suc(y) = leqN x y

addIn :: IN -> IN -> IN

addIn x zero = x

addIn Pos(x) Pos(y) = Pos(addN x y)

addIn Pos(x) Neg(y) = if (leq y x): Pos(sub x y)
else Neg(sub y x)

addIn Neg(x) Neg(y) = Neg(add x y)

addIn Neg(x) Pos(y) = if (leq y x): Neg(sub x y)
else Pos(sub y x)

2. Evaluate arithmetic expression

```
public static Pila plusPN(Pila exp)
{ Pila aux = new Pila(); Integer x, y;
  try { while (!exp.EsVacia()) {
      if (exp.Tope() instanceof Integer && !aux.EsVacia() && aux.Tope() instanceof Integer && aux.DesApila()).
        X = (Integer) aux.Tope(); aux = aux.DesApila();                    Tope.equals (new
        y = (Integer) exp.Tope(); exp = exp.DesApila();
        aux = aux.DesApila: exp.Apila(x+y);
    else { aux.Apila(exp.Tope()); exp = exp.DesApila(); {
    while (!aux.EsVacia()) }
        if( aux.Tope instanceof Character) exp.Apila(" + "));
        else } exp.Apila(Aux.Tope()); { aux = aux.DesApila; {
        return exp } call (TAD vacio exc e)} Sout ("Empty" +e); return exp;} }
```

3) a) lqueue [(CV: 2 : 3 : 1) : CV : (CV : 3 : 4)] = lqueue [(CV : 2 : 3 : 1) (CV : 3 : 4)] = [1 : 3 : 2 : 4 : 3]

b) HIGH LEVEL

```
public static <E> Lista<E> lqueue (Lista <Cola <E>> lq) throws TADVacioException
{ Lista <E> l = new Lista <E>();
  try { if (! lq.EsVacia())
        { Cola <> q1 = lq.Cabeza(); l = lqueue( lq.Cola());
          while (!q1.EsVacia())
            { l.Anade( q1.cabeza()); q1 = q1.Resto(); }
        }
      return 1; }
```

c) Haskell

```
lstack l :: [Pila a] -> [a]
lstack l    []   = []
lstack l    [ES : t] = lstack l t
lstack l    [x:q : t] = (lstack l [q]) ++ (x : (lstack t))
```

30th October 2019

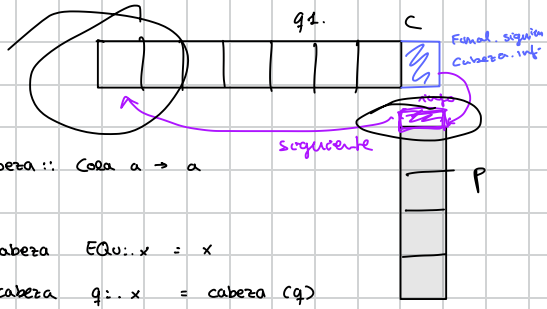
a)
```
leq Nl :: N1 -> N1 -> Bool
leq  One    Next x = True
leq  Next x  One   = False
leq  Next x  Next y = leqN1  x  y
```

```
leq Inte N1 :: InteN1 -> InteN1 -> Bool
leq Inte N1  None   ____  = True
leq Inte     None   Neg x = False
leq Inte N1  Neg x  ____  = True
leq Inte N1  Pos x  ____  = False
leq Inte N1  Pos x  Pos x = leq N1  x  y
leq Inte N1  Neg x  Neg x = leq N1  y  x
```

```
addN1 :: N1 -> N1 -> N1
addN1   x    One  = Next  x
addN1   x    Next y = add Next x  y
```

```
addInt :: inteN1 -> N1 -> InteN1
addInt  None    x   = Pos x
addInt  Pos x   y   = Pos addN1 (x  y)
addInt  Neg (One)  One  = None
addInt  Neg (One)  Next x = Pos  x
addInt  Neg (next x)  One = Neg (x)
addInt  Neg (next x)  (next y) = AddInt (neg x  y)
```

2)



a) Haskell rules:

octubre19:: Cola a → Pila a

octubre19 EQu = ES

octubre19 q:.x = x:/ (o19 (q:.x) ((cabeza (q:.x)):/N)

cabeza :: Cola a → a

cabeza EQu:.x = x

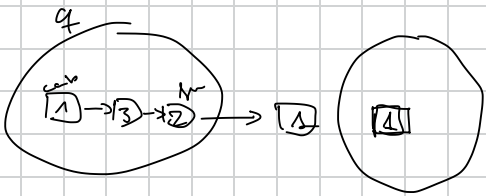
cabeza q:.x = cabeza (q)

o19:: Cola a → Pila a → Pila a

o19 EQu p = p

o19 q:.x p = o19 q x:/p

octubre19 (CV:/ 1:/ 3:/ 2) = 2:/1:/3:/2:/1:/ ES




```
public static <E> Pila<E> octubre19 (Cola <E> q)
{ Cola <E> q1 = new Cola<E> (q.clone()); Pila<E> p1 = new Pila_Dinamica<E> ();
  try { if (!q1.EsVacia)                                  E aux  P2
        { p1.Apila (q1.Cabeza());
          while (!q1.EsVacia())
           { p2.Apila (q1.Cabeza()); Aux = q1.Cabeza
             q1 = q1.Resto();
           }
          while (!p2.EsVacia())
           { p1.Apila (p1.Tope());
             p2 = p2.DesApila();
           }
          p1.Apila(Aux);
        } catch ....
    return p1
```

1) nfB3N :: BoolThreeNat → BoolThreeNat

nfB3N (B3N Zero Zero = B3N Zero Zero

nfB3N Suc x Zero = B3N Suc Zero Zero

nfB3N Zero Suc x = B3N Zero Suc Zero

nfB3N (B3N Suc x Suc y) = nf(B3N x y)

andB3N → Bool → Bool → Bool

andB3N B Zero Zero B Zero(Suc Zero) = Zero Suc Zero

andB3N B Zero(Suc Zero) _ = B3N Zero(Suc Zero)

andB3N _ _ = Zero Zero

andB3N Suc Zero Zero X = X

andB3N B3N x B3N y = andB3N (nfB3N B3N x) (nfB3N B3N y)

2) a) quevexam (EQu:. 3) = Equ :. 2 :. 1

quevexam (EQu:. 3:_ 2:.1) = Equ :. 3:. 1 :. 2

b) HIGH LEVEL

```
public static Cola<Integer> quevexam (Cola<Integer> q)
    { CD <Integer> sol = new CD <Integer> ();
      try { if( q. EsVacia || q. Resto(). EsVacia)
            { sol. Encola (2);
              sol. EnCola (1);
            }
          else{ int x, y;
                while (!q1. Resto(). Resto(). EsVacia()) {
                      sol. encola (q. Cabeza); q= q. Resto();}
                X = q. Cabeza(); q = q. Resto();
                sol. Encola (q. Cabeza) q = q. Resto()
                sol. Encola (x); }
                catch ( TAD VacioException exp) { exp sout "empty queue" }
      return sol
    }
```

LOW LEVEL

```
public void poneUltimo()
    { if (NodoCabeza == null || NodoCabeza.Siguiente == null)
        { NodoFinal = new Nodo (1, null);
          NodoCabeza = new Nodo (2, NodoFinal); }
      else {
          Nodo <E> aux = NodoCabeza
          while (aux.siguiente != NodoFinal)
                aux = aux.Siguiente();
          E x = aux.Info(); aux.Info = NodoFinal.Info; NodoFinal.Info = x }
```

October 6th 2021

```
nfBoolNand :: BoolNand → BoolNaand
nfBoolNand    yes  =  yes
nfBool Nand   Nand yes yes = Nand  yes yes
nf BoolNand   Nand yes  (Nand  yes yes) =  yes
nfBoolNand  (Nand ( Nand yes yes) Yes   =  yes
nfBoolNand  Nand (Nand yes yes)(Nand yes yes) = Yes
nf Bool Nand  (Nand  x      y ) = nfBoolNand   (Nand (nfBoolNand x)  (nfBoolNand y))
```

(2) a) 1 = EQu :. 3 :. 4 :. S 3 = (mcx21 (3 :/ 2 :/ ES EQu) :. 1 = = EQu :. 1

2 = EQu 4 = (EQu):. 4 :. S :. 1

b) HIGH LEVEL

```
public static <E> Cola <E> mcx21 ( Pila <E> s, Cola <E> q)
  {
     Cola q2 = new Cola <E> (q.clone());
       if (! s.EsVacia && ! q2.EsVacia())
           q2 = q2.Resto(); q2.EnCola ( s.Tope()); }
       return q2;
```

LOW LEVEL

```
public void mux2l (PDC<E> s)
    if ( NodoCabeza != null && s. NodoCabeza != null)
        NodoCabeza = NodoCabeza. Siguiente.
        NodoFinal. Siguiente = new Nodo <E> ( s. NodoCabeza. Info, null);
```

CONSTRUCTORS

→ No returns

→ Can combine LOW and HIGH level.

→ Dinámicas = LOW, interfaces = HIGH

HASKELL:

exam:: [a] → [a] → [a] (from constructor of the exam)

Haskell specification for JAVA CODE of trees:

```
part:: Arbol a → Arbol a → Arbol a
part  AV        AV    = AV
part  (AB r - -) AV   = AB r AV AV
part  -  (AB r - -)   = AB r AV AV
```

```
exam:: [a] → [a] → [a]
exam  []    l2 = l2
exam  x:l1  l2 = exam (l1  x:l2)
```

```
exam2:  [a] → [a] → [a]
exam2   EL    EL   = EL
exam2   EL    l    = l
exam2   l     _    = (reverse l) ++ _
reverse :: [a] → [a]
reverse  EL  = EL
reverse  (x:y:l) = (reverse (x:l) : y
```

Class:

```
aux:: [a] → [a] → [a]
aux  []   l  = l
aux  x:l2 l  = aux (l2) (x:l2)
```

Haskell specification for JAVA CODE of trees:

```
part :: Arbol a -> Arbol a -> Arbol a
part AV AV = AV
part (AB r - -) AV = AB r AV AV
part - (AB r - -) = AB r AV AV
```

```
exam :: [a] -> [a] -> [a]
exam [] l2 = l2
exam x:l1 l2 = exam (l1 x:l2)
```

```
Class:
aux :: [a] -> [a] -> [a]
aux [] l = l
aux x:l1 l2 = aux (l1) (x:l2)
```

```
exam2 :: [a] -> [a] -> [a]
exam2 EL EL = EL
exam2 EL l = l
exam2 l _ = (reverse l) ++ _
reverse :: [a] -> [a]
reverse EL = EL
reverse (x:y:l) = (reverse (x:l) : y
```

```
public ABD( ArbolBinario <E> a1, ABD<E> a2 ){
  if(!a1.EsVacio() || a2.raiz != null
  { if ( a2.raiz == null && !a1.EsVacio)
    { this.raiz = New NodoArbol<E>(a1.Raiz(), null, null);
    }else if ( a1.EsVacio && a2.raiz != null)
      { this.raiz = New NodArbol<E>(a2.raiz.Info, null, null); }
    else{
      if (a2.raiz.Info.CompareTo(a1.Raiz()) > 0)
        this.raiz = new NodoArbol<E>(a1.Raiz(), null, a2.raiz.Info);
      else if (a2.raiz.Info.CompareTo(a1.Raiz()) < 0)
        this.raiz = new NodoArbol<E>(a1.Raiz(), a2.raiz.Info, null); }
    }
  else { this.raiz = null; }
  catch (TAD Vacio Exception exp) { exp.printStackTrace(); }
}
```

Que el compareTo() de menor que 0

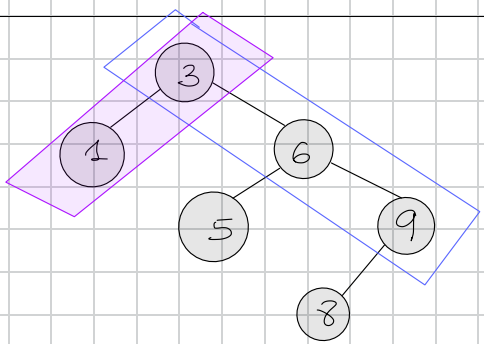
Significa que el objeto this utilizado es menor que el encontrado en los parentesis.

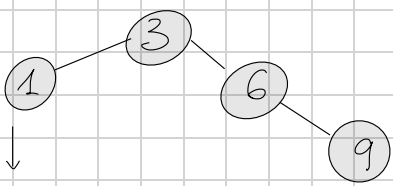
JAVA EXERCISE

If both have a root: the ArbolBinario = new root ; and if ABD< it will be left else right

HASSKELL

```
data Arbol a = AV | AB a (Arbol a) (Arbol a)
belong :: (Ord a) → a → Arbol a → Bool.
belong x AV = False
belong x (A r i d) = if | x == r = True
                        | x < r = belongs x i
                        | x > r = belongs x d
```

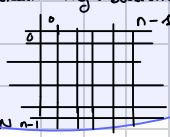




(AB 3 (AB 1 AV AV) (AB 6 AV (AB 1 AV AV))

```
mimax :: (Ord a) → Arbol a → Arbol a
mimax AV = AV
mimax (AB r i d) = AB r (aux True i) (aux False d)
aux :: Bool → Arbol a → Arbol a
aux _ AV = AV
aux True (AB r i d) = AB r (aux True i) AV
aux False (AB r i d) = AB r AV (aux False d)
```

```
public boolean mg(boolean m[][])
{ }
```



EXAM → GNDE: Nodos, Adyacencias, numnodos → ITERATION n-1

GRAPHS JAVA:
- LOW LEVEL → ADYACENCIA.
- HIGH LEVEL → Nodos(); Lista<E> adyacentes

METHOD PROPOSED WITH GRAPH will be something like → (Constructor) public GNDE (Lista<E> q) { }

SECOND PROGRESS TEST

1. SPECIFICATION:

a) dec22 :: Grafo 1 a → Grafo 1 a

i) dec22 (Nodo 1 (Arco 1 1 GVacio)) → dec22 (Arco 1 1 (Nodo 1 GVacio)) = dec22 (Nodo 1 GV)

= Nodo (1 GVacio)

ii) dec22 (Nodo 1 (Arco 2 1 (Nodo 2 GVacio))) → dec22 (Arco 2 1 (Nodo 1 (Nodo 2 GV))) =

dec22 Nodo 1 (Nodo 2 GV)) = Arco 1 2 (Nodo 2 (dec22 (Nodo 1 GV))

Arco 1 2 (Nodo 2 (Nodo 1 GV))

iii) dec22 (Nodo 1 (Arco 1 2 (Arco 2 3 (Nodo 2 (Arco 3 2 (Nodo 3 GV)))))) →

dec22 (Arco 1 2 (Nodo 1 (Arco (.....)) →

dec22 (Nodo 1 (Arco (y,z,g))

dec22 (Arco 2 3 (Nodo 1 (Nodo 2 (Arco 3 2 (Nodo 3 GV)))))

dec22 (Nodo 1 (Nodo 2 (Arco 3 2 (Nodo 3 GV)))) =

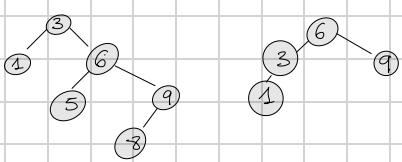
Arco 1 2 (Nodo 2 (dec22 (Nodo 1 (Arco 3 2 (Nodo 3 GV)))

Arco 1 2 (Nodo 2 (dec22 (Arco 3 2 (Nodo 1 (Nodo 3 GV))))

Arco 1 2 (Nodo 2 (dec22 (Nodo 1 (Nodo 3 GV))))

Arco 1 2 (Nodo 2 (Arco 1 3 (Nodo 3 (Nodo 1 GV))

b) mimax :: Arbol a → Grafo 1 a



mimax AV = GV

mimax (AB x i d) = aux False x d (mimax True x i (Nodo x GV))

aux :: Bool → a → Arbol a → Grafo 1 a → Grafo 1 a

aux _ _ AV g = g

aux True x (AB y i _) g = mimax True x i (Arco x y (Nodo y g))

aux False x (AB y _ d) g = mimax False x d (Arco x y (Nodo y g))

no debería ser y?

DUDA

mcmax (AB 3 (AB 1 AV AV) (AB 6 (AB 5 AV AV) (AB 9 (AB 8 AV AV) AV))) → 9

aux False 3 (AB 6 (AB 5 AV AV) (AB 9 (AB 8 AV AV) AV))) (aux 3 (AB 1 AV AV (Nodo 3 6V

aux False 3 (AB 9 (AB 8 AV AV) AV) Arco (3 6 (Nodo 6 (aux 3 (AB 1 AV AV (Nodo 3 6V)))

aux False 3 AV (Arco 3 9 (Nod 9 (Arco (3 6 (Nodo 6 (aux True 3 (AB 1 AV AV (Nodo 3 6V)

Arco (3 9 (Nodo 9 (Arco 3 6 (Nodo 6 (aux True 3 AV (Arco 3 1 (Nodo 1 (Nodo 3 6V))

Arco (3 9 (Nodo 9 (Arco 3 6 (Nodo 6 (Arco 3 1 (Nodo 1 (Nodo 3 6V)))))

EXAMEN HASKELL → NO GRAPHS

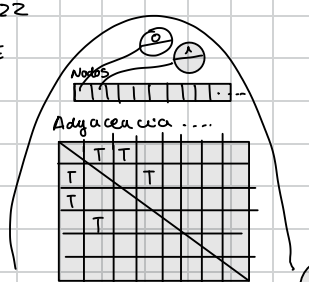
↳ NO GENERAL TREES

DIFFERENCE BETWEEN COMPLETE AND FULLY COMPLETE
- COMPLETE: Holes in last level
- FULLY COMPLETE: No holes.

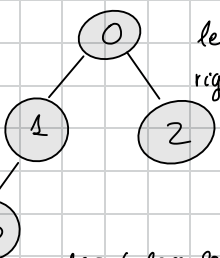
MANDATORY ITERATIVE

EXAM 2022

GNDE



n = 4



left: 2 · i + 1

right: 2 · i + 2

NomNodos = 4 (example)

preorder (from left to right)
[3, 1, 0, 2]

in order [0, 1, 2, 3]

Fully Complete

$n = 2^n - 1$

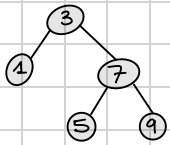
n° nodos = complete

variant n = height y create fully complete.

```
public GNDE (Integer n)
{   Adyacencias = new boolean [MAX_NODOS][MAX_NODOS];
    Nodos = new NodoGrafo [MAX_NODOS]; numnodos = 0;
    //Complete.
```

OTHER

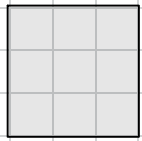
a) [1, 3, 5, 7, 9]

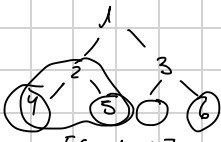


testAVL:: (Arbol a) → Bool

testAVL AV = True

testAVL (AB r i d) = if abs ((heigh i) − (heigh d)) > 1 then False.
else (test AVL i) && (test AVL d)





4 < 5

Subz = 4

[6, 1, 0]

EXAM December 13th 2022.

a) ArbolBinario.

Input → Complete tree. While

Left = 0 Right = 1.

public static <E> Lista<Integer> hole (ArbolBinario<E> t)

{Integer l, r; Lista<Integer> sol, left, right; sol = new LD<>();

if(t.EsVacio) sol.Añade(0);

else try {left = hole(t.subArbolIzqdo()); right = hole(t.subArbolDcho());

l = left.Cabeza(); r = right.Cabeza();

if (l<r)) { sol = left.Cola(); sol.Añade(0); }

else if (l>r) { sol = right.Cola(); sol.Añade(1); }

sol.Añade(l+r+1)

PREGUNTA

b) ITERATIVE CONSTRUCTOR

public GNDE (ArbolBinario<E> t)

{

try{ while (!t.EsVacio)

{ this.Raiz() =

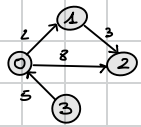
PRACTICES:

~~NO GRADE~~ (We only work with adyacency matrix → FLOYDS ALGORITHM

↑

No weighted (adyacency boolean)

No directed



adj

No arrow = ∞

	0	1	2	3	4	5	6
0	I	2	8	I			
1	I	I	3	I			
2	I	I	I	I			
3	5	I	I	I			
4					I		
5						I	
6							I

Mejor poner 0 en la diagonal.

* always arrow with a node and itself.

Weighted GRAPH

(Directed (symetrical adj matrix)

(Integers in adyacency)

path weigth

→ To go from 0 to 2 : [0, 2] → 8

[0, 1, 2] → 2+3 = 5

Integer ∞ = Integer. MAX_VALUE ~> ∞

best path → LOWEST WEIGHT

HIGHEST WEIGHT

adj

	2
0	5

best path LOWES WEIGHT.

path

	2
0	[]

→ [0, 1, 2]
[0, 2]

matrix of lists of paths.

```
lpc:: Arbol a → Int.
lpc   AV = 0
lpc AB_ AV AV = 1
lpc (AB_ i AV) = -1
lpc (AB_ AV d) = -1
lpc CAB_ i d) = if ((lpc i) == -1) || ((lpc d) == -1)   then = -1
                else (lpc i) + (lpc d)
```

```
lpc:: Arbol a -> Int
lpc  AV = 0
lpc (AB_ i d) = aux (lpc i) (lpc d)
```

```
aux:: Int → Int → Int
aux  li  ld = if (li == (-1)) || (ld == (-1))
              else  li + ld.
```

b) ITERATIVE → GNDE

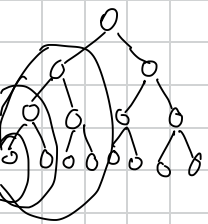
SEARCH TREE (mcmax)

```
public GNDE(ArbolBinario<E> t)        ITERATIVE → methods of ArbolBinario.
{ Adyacencias = new boolean[MAX_NODOS][MAX_NODOS];
  Nodos = new NodoGrafo[MAX_NODOS];
  numNodos = 0; ArbolBinario<E> aux;
  if(!t.EsVacio())
    try{ Nodos[0] = new NodoGrafo(t.Raiz(), false);
         numnodos++;  aux = t.SubArbolIzqdo();
         while(!aux.EsVacio())
         { Nodos[numnodos] = new NodoGrafo(aux.Raiz(), false);
           Adyacencias[numNodos][numnodos-1] = Adyacencias[numnodos-1][numnodos] = true.
           numnodos++;  aux = aux.SubArbolIzqdo();
         }
         if (!t.SubArbolDcho().EsVacio())
         { Nodos[numnodos] = new NodoGrafo(t.SubArbolDcho().Raiz(), false);
IMPORTANT → Adyacencias[numnodos][0] = Adyacencias[0][numnodos] = true;
           numnodos++;  aux = t.SubArbolDcho().SubArbolDcho();
           while(!aux.EsVacio())
           { Nodos[numnodos] = new NodoGrafo(aux.Raiz(), false);
             Adyacencias[numnodos][numnodos-1] = Adyacencias[numnodos-1][numnodos] = true;
             numnodos++;  aux = aux.SubArbolDcho();
           }
         }
         catch(TADVacioException ex) {ex.printStackTrace();}
```

Grafo

adyacencias

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						



30th November 2021.

1) No Graphs and no general trees.

2)
```
public GNDE (Integer n)
{ Adyacencias = new boolean [MAX_NODOS][MAX_NODOS];
  Nodos = new NodoGrafo [MAX_NODOS]; numnodos = 0;
  for(i=0; i<n; i++)
    { Nodos [i]= new NodoGrafo (i, false)}
  for(i=0; i<n; i++)
    { if(i*2+1 < n) Adyacencias [i][i*2+1] = Adyacencias [2*i+1][i] = true;
      else if(i*2+2<n) Adyacencias [i][i*2+2] = Adyacencias [2*i+2][i] = true;
      else break;
    }
}
```

18th December 2020

My way ITERATIVE

```
public Lista_Dinamica <E> FromLevel (int level)
{ int counter_level = 1; aux=new ArbolDinámico(); aux2= new ArbolDinamico (); sol= new ListaDinamica()
  if ( counter_level >= level )
    { sol.Añade (this.Raiz()); }
  try { while (!this. EsVacio());
    { counter_level ++; aux= this.SubArbolIzqdo(); aux2 =this.SubArbolDcho();
      if ( counter_level >= level )
        { sol. Añade( aux.Raiz()); sol. Añade (aux2. Raiz()); }
    }
  return sol }
```

Solution RECURSIVE

```
public Lista_Dinamica <E> FromLevel (int level)
{ Lista_Dinamica <E> sol = new Lista_Dinamica ();
  if(!EsVacio())
  { try { Lista_Dinamica aux = ((ArbolBinarioDinamico <E>) SubArbolDcho()). FromLevel (level-1));
    if( level <= 1) sol. Añade (Raiz());
    sol=(ArbolBinarioDinamico <E>) SubArbolIzqdo()). FromLevel (level-1);
    sol. Concatena (aux);
    } catch (TAD VacioException exp) { exp. printStackTrace());
    return sol;
  }
```

```
public GNDE (ArbolBinario<E> t)
  { Adyacencias = new boolean [MAX_NODOS][MAX_NODOS]; Nodos= new NodoGrafo [MAX_NODOS]; numnodos =0;
    if(!t.EsVacio())
      try { Nodos [0]= new NodoGrafo (t.Raiz(), false); numnodos= 1;
            aux20(t.SubArbolIzqdo, 0); aux2(t.SubArbolDcho, 0);
          } catch (TADVacioException exp ) { System.... (exp) }
  }

private void aux20 (ArbolBinario<E> t, int fat)
  { if (!t.EsVacio)
      try { Nodos [numnodos ]= new NodoGrafo ( t.Raiz(), false );
            Adyacencias [numnodos][ fat ] = Adyacencias [ fat][numnodos ] = true;
            fat= numnodos ; numnodos ++ ;
            aux20 (t.SubArbolIzqdo(), fat); aux (SubArbolDcho(), fat );
      } catch (TAD Vacio Exception exp ) { System.... ( exp); }
```

27th November 2019.

```
lemleaf :: Arbol6 a -> Int
lemleaf  AGV =  0
lemleaf  AG _ []= 1
lemleaf  AG _ [t:]= 1+ lemoleaf  t

rimoleaf :: Arbol6 a -> Int
rimoleaf  AGV = 0
rimoleaf  AG r []=1
rimoleaf  AG _ [ g ] =  1 + rimoleaf g
rimoleaf  AG r [ _ : t ]= rimoleaf (AG  r  t)

alleaves :: Arbol6 -> Bool
alleaves  AGV = True
alleaves  AG r [] = True
alleaves  g  =  auxal  (lemoleaf  g)  g

auxal  Int -> Arbol6  a -> Bool
auxal  n  AG _ []  =  if  n == 1  -> True    else  False
auxal  n  Ag  r [g] = auxal (n-1) g
auxal  n  Ag  r (g:t) = [auxal  (n- 1) g ]  && [ auxal  n (Ag  r  t) ]
```

```
public static Integer maximum (ArbolBinario<Integer> t) throws TADVExp
  { if (!t.EsVacio())
      try { if (t.SubArbolIzqdo.EsVacio()) return t.Raiz();
            else return maximum(t.SubArbolIzqdo());
          }

public void replaceroot (NodoArbol<Integer> t, Integer y) throws IllegalArgumentException
  { try { if(n.iz != null && (y <= (Integer) (maximum (new ArbolBinarioDinamico (n.iz))))
                                  y.compareTo(maximum(new ArbolBinarioDinamico(n.iz))) >= 0
            throw new IllegalArgumentException();
          if (n.de != null && (y >= (Integer)(minimum(new ArbolBinarioDinamico (n.de)))))
                                  y.compareTo(minimum(new ArbolBinarioDinamico(n.de))) <= 0
            throw new IllegalArgumentException();
    (else) n.valor = y;
        } catch (TADVacioException exp) { Sys... (exp); }
    }

public void replace (Integer x, Integer y) throws IllegalArgumentException
  { if (raiz == null) throw new IllegalArgumentException;
    else { NodoArbol<E> n; n= raiz;
      while(n != null && x != (Integer)(n.valor))
          { if (x < (Integer)(n.valor))
                  if (y >= (Integer)(n.valor))
                      throw new Illegal argu....
                  else n= n.iz;
            else if(y <= (Integer)(n.valor))
                  throws new IllegalArgumen...
            else
              n= n.de;
          }
      if (n == null) throw new IllegalArgument....
      else replaceroot(NodoArbol<Integer> n, y);
  }
}
```

REPASO TODOS EXAMENES

HASSKELL

```
mcmax :: (Ord a) -> Arbol a -> Arbol a
mcmax AV = AV
mcmax (AB r i d) = AB r (aux True i) (aux False d)
aux :: Bool -> Arbol a -> Arbol a
aux _ AV = AV
aux True (AB r i d) = AB r (aux True i) AV
aux False (AB r i d) = AB r AV (aux False d)
```

```
mcmax:: (Ord a) Arbol a -> Arbol a
mcmax AV = AV
mcmax AB r AV AV = AB r
mcmax AB r i d = AB r (aux True i) (aux false d)
aux:: Bool->Arbol a->Arbol a
aux _ AV = AV
aux True AB r i d = AB r (aux True i) AV
aux False AB r i d = AB r AV (aux False d)
```

NOVEMBER 30th 2021

2) public static <E> ArbolBinario <E> prune (Lista ag)

```
{ ArbolBinario<E> sol = new ABD<E>();
  if(!ag.EsVacia)
  try { ArbolBinario<E> left = new ABD<E>();
                        right = new ABD<E>();  E root = (E) ag.Cabeza(); ag = ag.Cola();
        while (!ag.EsVacio && !ag.Cola().EsVacia() && !ag.Cola().Cola().EsVacia())
          { ag = ag.Cola(); }
          if(!ag.EsVacia)
             { left = prune ((lista) ag.Cabeza()); ag = ag.Cola();
               if (!ag.EsVacia) right = prune((lista) ag.Cabeza());}
             sol = new ABD<E> (root, left, right);
    } catch (TADVacioException exp){ Sout (exp)}
  return sol;
```

[[ ], [ ], [ ], [ ], [ ]]

b) public GNDE (Integer n)

```
{ Adyacencias = new boolean [MAX_NODOS][MAX_NODOS];
  Nodos = new NodoGrafo [MAX_NODOS]; numnodos = 0; Integer i; numnodos = n;
  for (i=0; i< n; i++) Nodos[i] = new NodoGrafo (i, false)
  for(i=0; i<=n; i++)
     if( i*2+1 < n) Adyacencias [i][i*2+1] = Adyacencias [i*2+1][i] = True ; else break;
      if(i*2+2 < n) Adyacencias [i][i*2+1] = Adyacencias [i*2+2][i] = True. else break;
   }
{
```

December 13th, 2022

```
mcmax :: Arbol a → Arbol a
mcmax AV = AV
mcmax AB r i d = AB r (aux True i) (aux False d)
aux :: Bool → Arbol a → Arbol a
aux _ AV = AV
aux True (AB r i d) = AB r (aux True i) AV
aux False (AB r i d) = AB r AV (aux False d)
```

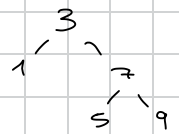
2. IMPLEMENTATION

```
public static <E> Lista<Integer> hole (ArbolBinario<E> t)
  { Integer l, r; Lista<Integer> sol, left, right; sol = new LD<>();
    if (t.EsVacio()) sol.Añade(0);
    else
      { try { left = hole(t.SubArbolIzqdo());
              right = hole(t.SubArbolDcho());
              l = left.Cabeza(); r = right.Cabeza();
              if (l<r) { sol = left.Colac(); sol.Añade(0); }
              if (l>r) { sol = right.Colac(); sol.Añade(1); }
              sol.Añade(l+r+1);
        } catch (TAD...){}
        Return sol;
      }
```

```
public GNDE (ArbolBinario<E> t)
  { Adyacencias = new boolean[MAX_NODOS][MAX_NODOS];
    Nodos = new NodoGrafo[MAX_NODOS]; numnodos = 0;
    ArbolBinario<E> aux;
    if (!t.EsVacio())
    { Nodos[0] = new NodoGrafo(t.Raiz(), false); numnodos++;
      aux = t.SubArbolIzqdo();
      while (!aux.EsVacio())
      { Nodos[numnodos] = new NodoGrafo(aux.Raiz(), false);
        Adyacencias[numnodos][numnodos-1] = Adyacencias[numnodos-1][numnodos] = true; numnodos++
        aux = aux.SubArbolIzqdo();
      }
      if (!t.SubArbolDcho.EsVacio())
      Nodos[numnodos] = new NodoGrafo(t.SubRa, false);
      Adyacencias[0][numn...] = Adyacencias[numn...][0] = true;
      numnodos++; aux = t.SubDcho().SubDcho();
      while (!aux.EsVacio)
      { Nodos[numnodos] = NodoGrafo(aux.Raiz(), false);
        Adyacencias[numnodos][numnodos-1] = Adyacencias[numnodos-1][numnodos] = true;
        numnodos++;
        aux = aux.SubArbolIzqdo();
      } catch (VA...)
```

December 18th 2020

2)

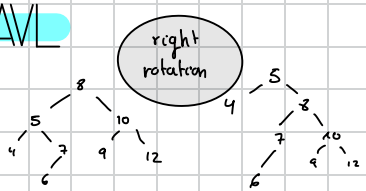


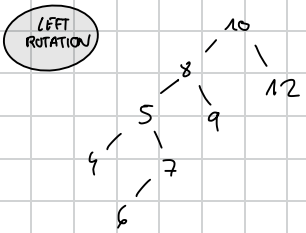

```
public Lista_Dinamica <E> FromLevel (int level)
  { Lista_Dinamica <E> sol = new Lista_Dinamica <E> ();
    if(!EsVacio)
      {try { List_Dinamica <E> aux = ((ArbolBinario)(SubArbolDch.FromLevel(level-1));
             if (level <= 1)  aux.Añade(Raiz());
               sol = (ArbolBinario) SubarbolIzqdo.FromLevel (level - 1)
               sol.Concatena(aux);
       { catch (TADVacio Excepcion) { e.printStackTrace(); }
      return sol;
```

b) public GNDE (ArbolBinario<E> t)

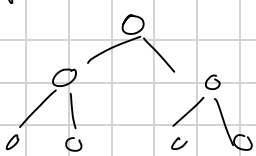
```
  { Adyacencias = new boolean[MAX_NODOS][MAX_NODOS];
    Nodos = new NodoGrafo[MAX_NODOS]; numnodos = 0;
    if(!t.Esvacio)
      {try { Nodos[0] = new NodoGrafo(t.Raiz(), false); numnodos++;
           while(!t.EsVacio)
             aux(t.SubArbolIzqdo, 0); aux(t.SubArbolDcho, 0);
         }
      catch (TADVacio Excepcion exp) (Sout(exp)) }

  private void aux (ArbolBinario<E> t, int fat)
     { while (!t.EsVacio)
         { try { Nodos[num] = new NodoGrafo(t.Raiz(), false);
                 Adyacencias[fat][numnodos] = Adyacencias[numnodos, fat];
                  fat = numnodos; numnodos++;
                  aux 20 (t.SubArbolIzqdo(), fat); aux 20 (SubArbol Dcho(), fat);
             { catch (TAD Vacio Excepcion exp) ( Sout (exp))
         {
```

AVL





```
public static Integer minimum (ArbolBinario<Integer> t) throws TADVacioException
{ try{ while (!t.EsVacio)
ITERATIVE        {t.SubArbolIzqdo(); }
       {catch (TADVacioException exp) { Sout(exp)}
   return t.Raiz();

RECURSIVO  public static Integer minimum (ArbolBinario<Integer> t) throws TADVacioException
   { if (t.SubArbolIzquierdo().EsVacio()) return t.Raiz();
     else return minimum(t.SubArbolIzqdo());
   {
```



```
public void replaceroot (NodoArbol<Integer> n, Integer y) throws IllegalArgumentException
   { if( n == null) throw IllegalArgumentException;
     else if (n.iz >= y || n.de <= y || y == n) throw IllegalArgumentException.
                  comprueba maximos
     else
     {n.valor = y } catch TAD...)        y <= (Integer)(maximum)(new ArbolBinarioDinamico(n.iz)))
                                         y >= (Integer)(minimum(new ArbolBinarioDinamico(n.de)))

public void replace (Integer x, Integer y) throws IllegalArgument Excepton.
   { if (raiz = null) thrau Ill...Arg....;
     else { nodoArbol<E> n; n=raiz;
          while(n!=null && x!=(Integer) n.valor)
             { if(x<((Integer) n.valor)
                    if(y>=(Integer) n.valor) throw Illeger...)
                    else n=n.iz;
               else
                    if(y<=((Integer) n.valor) throw Illegal....;
                    else n=n.de;
             { if(n==nu ) throw...
               else replace root (n, y)
```

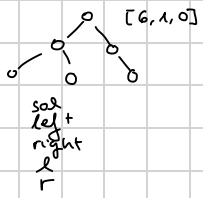
```
public GNDE (ArbolBinario<E> t) {
    Adyacencias = new boolean[M_N][M_N]; Nodos = new NodoGrafo[M_N];
    numnodos = 0;
    if(!t.EsVacio())
      { try { Nodos[0] = new NodoGrafo (t.Raiz(), false); numnodos = 1;
              aux (t.SubArbolIzqdo, 0); aux (t.SubArbolDcho, 0);
        } catch____
```

```
private void aux (ArbolBinario<E> t, Integer i)
  { i (!t.EsVacio)
    { try { Nodos[numnodos] = new NodoGrafo (t.Raiz(), false);
            Adyacencias[numnodos][fat] = Ady[fat][numnodos] = true;
            fat = numnodos; numnodos++;
            aux 20 (t.SubArbolIzq(), fat); aux (t.SubarbolDcho(), fat);
      { catch (...)
```

ITERATIVE

```
public GNDE (Integer n)
  { Ady = new bool [N][N]; Nodos = new NodoGrafo[N]; numnodos=0; Integer i;
    for (i=0; i<n; i++) Nodos[i] = new Nodo (i, false); }
    for(i=0; i<n; i++)
      | A
```

```
public static <E> Lista<Integer> hoje (ArbolBinario<E> t)
  { Integer l, r; Lista<Integer> sol, left, right; sol= new LD<>();
  if(!t.EsVacio){ try { left = hoje( t.SubArbolIzqdo ); right = (t.SubArbolDcho);
                l= (Integer) left.Cabeza(); r= (Integer) right.Cabeza();
                if (l< r) { sol = left.Cola(); sol.Anade(0); }
                if ( l> r) { sol = right.Cola(); sol.Anade(0); }
                 sol.Añade(l + r + 1);
                 } catch (TAD ___) { }
          }
    else if( t.EsVacio) sol. añade(0);
   }
  return sol;
```

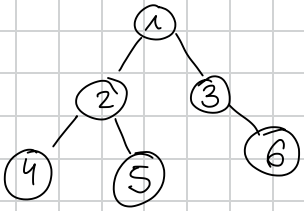



```
public GUDE (ArbolBinario<E> t)
  { Nodos = new NodoGrafo [MN]; Adyacencias = new boolean [MN][MN]; numnodos = 0; ArbolBinario<E> aux;
    if(!t.EsVacio)
    {try } Nodos [numnodos] = new NodoGrafo ( t.Raiz(), false);
           numnodos ++; aux = t.SubArbolIzqdo();
           while (!aux.EsVacio())
             { Nodos[numnodos] = new NodoGrafo ( aux.Raiz(), false);
               Adyacencias [numnodos][numnodos -1] = Adyacencias [numnodos -1][numnodos] = true;
               numnodos ++; aux = aux.SubArbolIzqdo();
             }
           if(!t.SubArbolDcho.EsVacio())
             {
```

```
hola :: Arbol a -> Bool
hola AV = False
hola AB _ AV AV = True
hola AB _ AV d = True && hola d
hola AB _ i AV = True && hola i
```

```
otro :: Arbol a -> AVL
otro AV = AV
otro AB _ AV AV =
```

```
aux :: Arbol a -> Arbol a -> int
aux AV AV = 0
aux AB r
aux AB
```



PRACTICE 3

```
public static Lista<ABD<Integer>> insert(ABD<Integer> x, Lista<ABD<Integer>> y) throws TADVacioException
    ArbolBonario aux = new ABD<Integer>();
{ try{ if(y.EsVacia()) y.Añade(x);
    else if(x.Raiz() <= y.Cabeza().Raiz()) y.Añade(x);
    else{ ABD<Integer> c;
          c = y.Cabeza();
          y = y.Cola();
          y = insert(x, y); y.Añade(c);}
          return y;
          }

public static ABD<Integer> huffmanTree (Lista<ABD<Integer>> a)
    sol = new ABD();
{ try{ if(a.EsVacia()){ return sol; }
    else if(a.Cola().EsVacia()) sol = a.Cabeza(); a = a.Cola();
    else{ while(a.Cola().EsVacia())
          {
            ABD aux = new ABD(a.Cabeza().Raiz() + a.Cola().Cabeza().Raiz(), a.Cabeza(), a.Cola.Cabeza());
            a = a.Cola().Cola();
        a = insert(aux, a);
            {
            Sol = a.Cabeza();
            { catch...
            return sol;
      {
```

PRACTICE 4

FLOYDS ALGORITHM:

```
public static Lista<Integer>[][] Floyd(int[][] adj)
{ Lista<Integer>[][] path = new LD[adj.length][adj.length];
    if(!adj.length!=0)
        for(i=0; i<path.length; i++)
            for(j=0; j<path.length; j++)
                path[i][j]= new LD<>();
                if(i!=j) && (adj[i][j]!= integer.MAX_VALUE)
                    { path[i][j].Añade(i); path[i][j].Añade(j); }}}
// FLOYD ALGORITHM
    for(k=0; k<path.length; k++)
        for(i=0; i<path.length; i++)
            for(j=0; j<path.length; j++)
                if( adj[i][k]!= Integer.MAX_VALUE); && (adj[k][j] != Integer.MAX_VALUE) && ( adj[i][k] +
                                                                                              adj[k][j] < adj[i][j]
                    try{
                        adj[i][j] = adj[i][k] + adj[k][j];
                        path[i][j] = (LD<Integer>) path[i][k].clone();
                        path[i][j].Concatena(path[j][k].Cola());
                    { catch (TADV___ ex) sout(exp)


public static Lista<Integer> cycle(int[][] adj)
    int inf = Integer.MAX_VALUE, minweight= Integer.MAX_VALUE, n= adj.length, i=0, j=0, k=0;
    Lista<Integer> sol= new LD<Integer>();
    Lista<Integer>[][] path = Floyd(adj);
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            if(i!=j && adj[i][j]!= inf && adj[j][i]!= inf && (adj[i][j]+adj[j][i] < minweight)
                minweight = adj[i][j] + adj[j][i];
                sol=(Lista<Integer>)(path[i][j].Cola().clone());
                sol.Concatena(path[j][i].Cola());}
            return sol;
```

PRACTICE EXAM

20th December 2023

2 + ((12 % 5) * 3) en polish notation: + 2 (* (% 12 5) 3) = + 2 * % 12 5 3

```
Pila s = new PD();   s.Apila(3); s.Apila(5); s.Apila(12); s.Apila("%"); s.Apila("*"); s.Apila(2); s.Apila("+");

public static Double PAValue (Pila exp)
  { Pila aux = new PD(); Double x, y; String op;
    try { while (!exp.EsVacia())
          { if (exp.Tope() instanceof Char)  Aux.Apila(exp.Tope());
            else if (exp.Tope() instanceOf Double && aux.Tope() instanceOf Double)
                 { x = aux.Tope(); y = exp.Tope(); aux = aux.DesApila(); (op = aux.Tope();)
                   aux.Apila(eval(x, (op), y);
                 }                                  (String)
                                                    aux.Tope()
            else { aux.Apila(Exp.Tope()); {
                   aux.Apila(exp Tope)
                   exp = exp.DesApila();
          } catch (TADVacioException) { System.__ }
          return (Double) exp.Tope();
    }
```

exp		aux															
2																	
*		*							12								
%		%		%			0/0		%				2				
12		~~12~~		~~12~~	*	12	*		*		*		*				
5		5	2	5	2	5	2	5	2	2	2		2		2		
3	+	3	+	3	+	3	+	3	+	3	+	3	+	6	+	8	

2. [Huffman trees]

```
public static int percent (ArbolBinario<Integer> t, Lista<Integer> l)
  { Integer p = -1; ArbolBinario<Integer> aux = new ABD<>();
    if (!t.EsVacio())
      try { if (l.EsVacia())
              if (t.SubArbolDcho().EsVacio()) p = t.Raiz(); }
            else if { l.Cabeza() == 0) p = percent(t.SubArbolIzqdo(), l.Cola());
                 else p = percent(t.SubArbolDcho(), l.Cola())
      } catch ___
    return p;
```

```
public static Lista<Integer> cycle (int[][] adj)
{ int inf = Integer.MAX_VALUE, minweight = Integer.MAX_VALUE, n = adj.length, i=0, j=0, k=0;
  Lista<Integer> sol = new LD<Integer>()
  Lista<Integer>[][] path = Floyd (adj);
    for (int i=0; i<n; i++)
      { for(j=0; j<n; j++)
          if (i!=j && adj[i][j]!=inf && adj[j][i]!=inf && adj[i][j]+adj[j][i] < minweight)
            { minweight= adj[i][j] + adj[j][i]; sol = (Lista<Integer>) (path[i][j].Copia()).clone();
              sol.Concatena(path[j][i].Copia());}
              return sol;
             {
```

FLOYD'S

```
public static int[][] transform (int[][] c)
  { int i, j, k; int inf = Integer.MAX_VALUE; minweight = Integer.MAX_VALUE; n = adj.length;
    int[][] adj = new int[c.length][c.length]; int weight = 1;
      for (i=0; i<c.length; i++)
          for(j=0; j<c.length; j++)
              k = c[i][j];
    if (i==j || k!=j && c[i][k] == k && c[k][j] == j)
          adj[i][j] = -1;
     else adj[i][j] = weight++;

public static int penultimate (int[][] c, int i, int j)
   int sol;
   if (i == j) sol = -1;
   else if ( c[i][j] == -1) sol = -1;
   else if ( c[i][j] == j) sol = i;
   else return penultimate ( c, c[i][j], j );
```

15th December 2021:

1) [Reverse Polish Notation]

```
public static Pila partialRPN (Pila exp, int i)
  Pila aux = new PD<>(); Double x, y; int counter;
  try{ while (!exp.EsVacia() && (i>0))
        if(exp.Tope() instance of Double) aux.Apila(exp.Tope());
        else { x = (Double) aux.Tope(); aux = aux.DesApila();
               y = (Double) aux.Tope(); aux = aux.DesApila();
               aux.Apila(eval(x, (String) exp.Tope(), y)); i--;
        exp= exp.DesApila();
      { while(!aux.EsVacia())
          { exp.Apila(aux.Tope()); aux = aux.DesApila();
        { return exp.
```

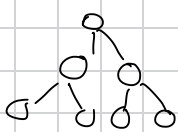
2) comhuff :: Int → Arbol Float int i < h < 16 (h: height)

```
public static ArbolBinario<Float> comhuff (int h)
  { return auxch(h, new Float(100), 1);
  }
public static ArbolBinario<Float> auxchr (int level, float perc, int ascii)
  { ABD<Float> ot = new ABD<Float>();
    if(level > 2) { ot = new ABD<Float>(perc, auxch(level-1, perc/2, ascii*2), auxch(level-1, perc/2, ascii+1));
    else ot = new ABD<Float>(perc, new ABD<Float>(new Float(ascii), new ABD<Float>(), new ABD<Float>());
    { return ot.
```

3) public static int[][] transform (int[][] c)

HUFFMAN TREE

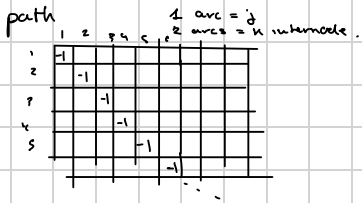
```
fox21:: ArbolFloat -> ArbolFloat
fox21 AV = AV
fox 21 AB _ n AV = AB 100 n AV
fix 21 AB p (AB _ n AV) (AB _ m AV) = AB p (P/2 n AV) (AB (P/2) m AV)
fox21 (AB p (AB _ n AV)(AB r i d)= AB p (p-r n AV) fox21 (AB r i d)
fix 21 (AB p (AB r i d)(AB _ n AV) = AB p fox21 (AB r i d) (AB p-r n AV)
fix 21 AB r i d = AB r fox21 i fox21 d

e19:: Arbol Int
e19 AB.

public static Integer RPNmaxbasec(Pila exp) { PD<> aux = new PD<>(); Double x, y;
  { if (!exp.EsVacia())
    try { while(!exp.Vacia())
          x = exp.Tope(); exp = exp.DesaPila();
          if( x instanceof Double && !exp.EsVacia() && exp.Despila.Tope instanceof Character)
              z = eval ( x, exp.Despila(). Tope(), exp.Tope());
              exp = exp.DesApila().Desapila(); if(z> max) max = z
```



HUFFMAN

```
public static ArbolBinario<Integer> mirror( ArbolBinario<Integer> h)
  { ArbolBinario<Integer> s;
    try { if(h.SubArbolDcho.EsVacio())
            s= (ArbolBinario<Integer>) h.clone();
          else s= new ABD<Integer>(h.Raiz(), mirror(h.SubArbolDcho().clone()),mirror:
```

```
e19:: Arbol a -> Arbol a
e19 AV = AV
e19 AB r i d = mirror AB
mirror: Arbol Int -> Arbol Int
mirror AB(r x AV)= AB r x AV
mirror AB(r i d) = AB r mirror d
                       mirror i
```



3) public static Pila<Integer> intermedios (int[][] c)

{ Pila<Integer> sal = new PD<Integer>(c);

int i, j, k, n = c.length;

for (i = 0; i < n; i++)

for (j = 0; j < n; j++)

{ k = c[i][j];

if { k != -1 && k != j && !Pertenece(sal, k)) sal.Apilar(k)

return sal

public static boolean Pertenece (PD<Integer> p, Integer k)

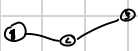
{ try { if (p.EsVacia()) return false;

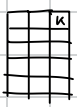
else return k == p.Tope() || Pertenece (p.DesApilar, k)

- 1 → no path
j → direct
k → conector

FLOYD PATH

1) public static int penultimate (int [][] c, int i, int j) int sal;





try { if (!c.EsVacia())

{ if (c [i][j] == -1) return -1

else if (c[i][j] == j) return i;

else return penultimate (c, c[i][j], j)

2) fx 21:: Arbol Float → Arbol Float

fx 21 AV = AV

fx AB _ n AV = AB 100 n AV

public static int[][] transform (int[][] c)

int[][] adj = new int[c.length][c.length]; int weight = 0;

for (int i = 0; i < c.length; i++)

for (int j = 0; j < c.length; j++)

{ if (c[i][j] == -1) adj[i][j] = -1

else if (c[i][j] == j) adj[i][j] = weight++;

else if (i == j || c[i][k] = k && c[k][j] == j && k != -1 && k != j)

adj[i][j] = -1;

return adj.

HUFFMAN TREE:

2) comhuff:: Int → Arbol Float

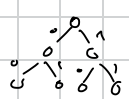
comhuff 0 = AV

comhuff h = auxhuff h 100 1

auxhuff :: Int → Float → Float → Arbol Float

auxhuff level perc ascii = if (level > 2) then AB perc (auxhuff level-1 p/2 ascii*2) (auxhuff level-1 p/2 ascii*2+1) else AB perc (AB ascii AV AV) AV

```
public static ArbolBinario<Float> comhuff (int h)
    return auxop(h, 100, 1);
}

public static ArbolBinario<Float> auxop ( level, perc, ascii)
    out = new ABD<Float>();
    if ( level > 2)
        out = new ABD ( perc, auxop(level - 1, perc/2, ascii * 2), auxop(level - 1, perc/2, ascii*2 +1)
    else out = new ABD (perc, new ABD( ascii, new ABD<Float>, new ABD<Float>)

public static Lista<Integer> cycle ( int[][] adj)
    { int inf = Integer.MAX_VALUE, minweight = Integer.MAX_VALUE, n = adj.length, i=0, j=0, k=0;
      Lista<Integer> sol = new LD<Integer>(); Lista<Integer>[][] path = Floyd (adj);
      for(int i=0; i<n; i++)
          for( int j=0; j<n; j++)
              if( i!=j && adj[i][j]!=inf && adj[j][i]!=inf && adj[i][j] + adj[j][i] < minweight)
                  minweight = adj[i][j] + adj[j][i];
                  sol (Lista<Integer>) path[i][j].Colar().clone();
                  sol.Concatenar (path[j][i].Colar().clone());
```



```
public static int percent (ArbolBinario<Integer> t, Lista<Integer> l)
    { Integer p = -1; ArbolBinario<Integer> aux = new ABD<>();
      if (!t.EsVacio())
          try{ if (!l.EsVacia())
                  { if( t.SubArbolDcho().EsVacio()) p= t.Raiz();}
              else if (l.Cabeza() == 0)  p= percent (t.SubArbolIzqdo(), l.Colar())
              else p.pe
```