

Y espe PILA [ELEMENTO]
usa Booleanos, naturales
Parámetro formal
genero elemento
f_paranetro
genero pila

Operaciones
ver : elemento, p:pila
contar : pila \rightarrow natural
fondo : pila \rightarrow elemento
montar : pila pila \rightarrow pila
quitar : pila natural \rightarrow pila

ecuaciones

fum contar (p:pila) : natural

Si^o pila_vacia?(P) entonces
Devolver 0
Sino

Devolver 1 + contar (desapilar(P))

f_fum

f_fum fondo (P:pila) : Elemento

Si^o pila_vacia?(P) Entonces
Error (Pila vacia)
Sino

e = cima (P)

desapilar(P)

Si^o pila_vacia?(P) entonces
Devolver e
Sino

fso fondo (P)

f_fum

fum montar (P1:pila, P2:pila) : pila

P = pila_vacia

Si^o pila_vacia?(P2) entonces

Devolver P2

Sino

mientras ! (pila_vacia?(P2)) entonces

apilar (desapilar(P2), P)

f_mientras

mientras ! (pila_vacia?(P)) entonces

apilar (desapilar(P), P1)

f_mientras

Devolver P1

f_fum

$$\begin{pmatrix} 3 \\ 2 \end{pmatrix} \begin{pmatrix} 4 \\ 6 \end{pmatrix} = \begin{pmatrix} 6 \\ 3 \\ 2 \end{pmatrix}$$

fun quitar(p_i: pila, n_o natural): pila

si pila_vacia?(p_i) entonces

Error (Pila vacia)

Sino

mientras n > 0 hacer

desapilar (p_i)

mientras n = n - 1

si:

fin

fspec

2) espec PILA [ELEMENTOS]

usa Booleanos, Naturales

por dentro poner

genero elemento

spacio

genero pila

es

Operaciones

contar? pila \rightarrow Natural

Inverso: pila \rightarrow pila

orden Mayor?: pila \rightarrow bool { Mayor fondo, Menor cima }

orden Menor?: pila \rightarrow bool

eliminar_fondo?: pila \rightarrow pila

ecuaciones

fun inverso (p_i: pila): pila

p: pila

si pila_vacia?(p_i) entonces

Devolver p_i

Sino

mientras !pila_vacia(p_i) entonces

apilar (desapilar(p_i), p)

mientras

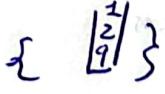


Devolver p

si:

fin

fun orden_Mayor? (p_i: pila): bool



si pila_vacia?(p_i) entonces

Error (Pila vacia)

Sino

e = cima(p_i)

desapilar (p_i)

b = true

mientras !pila_vacia(p_i) entonces

si ($- <_-(e, \text{cima}(p_i))$) entonces

e = cima(p_i)

desapilar (p_i)

Si no
b = False

fin

Mientras

Devolver b

fin si

fin

fun orden_Menor? (P: pila): bool

si pila_vacia?(P) entonces

Error (Pila vacia)

Sí no

e = cima(P)

desapilar(P)

b = true

mientras ! Pila_vacia(P) entonces

si ($- \leq -$ (cima(P), e)) entonces

e = cima(P)

desapilar(P)

Sí no

b = False

fin si

Mientras

Devolver b

fin si

fun eliminar_fondo(p: pila): pila

si Pila_vacia?(p) entonces

Error (Pila vacia)

Sí no

p = inverso(p)

desapilar(p)

devolver p

fin si

3) espec PILA[ENTEROS]

usa NATURALES

Parámetro formal

géreros Entero

Parámetro

géreros pila

Operaciones

sacar_en_pos: Pila natural \rightarrow Pila $\{ z : \text{cima}, z : \text{resto} \}$

sacar_entre: Pila natural natural \rightarrow Pila $\{ z : \text{resto} \}$

Ecuaciones

fun parcial sacar_en_pos(p:pila, m:natural): pila

Si^o pila_vacia?(P) entonces

Error (Pila vacia)

Símo Si^o (m > contar(P)) V (m ≤ 0) entonces

Error (Indice fuera de rango)

Símo

$$P_1 = \emptyset$$

$$m_1 = 1$$

mientras ! Igual?(m₁, m) entonces

$$m_1 = \text{succ}(m_1)$$

$$e = \text{cima}(P_1)$$

apilar(e, P₁)

desapilar(p)

f mientras

desapilar(P) {el que me interesa}

$$m_1 = 1$$

mientras ! igual?(m₁, m) entonces

$$m_1 = \text{succ}(m_1)$$

$$e = \text{cima}(P_1)$$

desapilar(P₁)

apilar(e, P)

f mientras

Devolver(P)

f5o

ffun

fun parcial sacar_entre(p:pila, m₁, m₂:natural): pila

Si^o pila_vacia?(P) entonces

Error (Pila vacia)

Símo

Si^o (m₁ < 0) V (m₂ < 0) V (contar(P) < m₂) V (m₁ > m₂) entonces { (P, 3, e₁) }

Error (Indice fuera de rango)

Símo

$$P_1 = \emptyset$$

$$e = \emptyset$$

mientras ! Igual(e, m₁) entonces

e = cima(P) → apilar(e, P₁)

$$e = \text{succ}(e)$$

f mientras

mientras ! Igual(e, m₂) entonces

desapilar(P)

$$e = \text{succ}(e)$$

f mientras

$$i = 0$$

mientras ! Igual(i, m₁) entonces

$$e = \text{cima}(P_1)$$

desapilar(P₁)

apilar(e, P)

f mientras

$$i = \text{succ}(i)$$

fsc
fso
ffun
fespe

mod - (fig. 9) ambas sentencias devuelven el resultado (9) para cada 38, 39, 40 y 41 en la lista de resultados.

4) espec PILAS [CONJUNTOS [ELEMENTOS]]

USA BOOLEANOS, NATURALES, conjuntos
parámetro formal

generos conjunto [elementos]

fparametros

generos pilas

operaciones

comprobar_elemento: elemento pila \rightarrow bool {Si esto es todos los conjuntos}

todos_elementos_en_pila: pila \rightarrow bool

combinar: pila \rightarrow pila

eliminar: pila \rightarrow conjunto

var_elemento_vacio: pila \rightarrow pila

ecuaciones: Paux : pila, Caux : conjunto

fun comprobar_elemento (e: elemento, p: pila): bool

Si es vacia? (p) entonces

Devolver False

Sino b_global = True b_parcial = False

mientras ! es vacia? (p) entonces

Caux = Cima (p)

mientras ! es vacia (Caux) entonces

Si es igual? (e, coger (Caux)) entonces

b_parcial = True

fsc

Borrar (coger (Caux), Caux)

mientras

Si b_global = F \wedge b_parcial = T entonces

b_global = T

fsc

desapilar (p)

fparametros

Devolver b_global

ffun

fun todos_n_comienem_cima(p:pila): boole

se esVacia?(p) entonces

Devolver true

sino

m-primer: motorop

c-aux = cima(p)

mientras ! esVacia(c-aux) entonces

suc(m-primer)

Borrar(coger(c-aux), c-aux)

m-estras

desapilar(p)

b-aux = true

mientras ! esVacia(p) entonces

m-conjunto: motorop

m-conjunto = 0

c-aux = cima(p)

mientras ! esVacia(c-aux) entonces

suc(m-conjunto)

Borrar(coger(c-aux), c-aux)

mientras

se ! es igual(m-primer, m-conjunto) entonces

b-aux = False

psf

f-mientras

Devolver b-aux

psf

end fun

quiero - elemento(p:pila): pila

se esVacia(p) entonces

error (pila vacia)

sino

p-aux: pila

mientras ! esVacia(p) entonces

c-aux = cima(p)

desapilar(p)

se ! esVacia(c-aux) entonces

Borrar(coger(c-aux), c-aux)

Apilar(c-aux, p-aux)

mientras

Devolver p-aux

psf

end fun

fun combinar (P:pil): conjunto

C-aux: conjunto

C-aux: conjunto

Si es_vacio(P) entonces

Devolver C-aux

Sino

C-aux2: conjunto

Mientras !es_vacio(P) entonces

C-aux2 = cima (P)

desapilar (P)

mientras !es_vacia(C-aux2) entonces

e: elemento

e = cojer (C-aux2)

Borrar (e, C-aux2)

Insertar (e, C-aux)

mientras

fmiembros

Devolver C-aux

fin

fun eliminar_conjuntos_vacios (P:pila): pila

Si es_vacio (P) entonces

Devolver P

Sino

Mientras !es_vacio (P) entonces

C-aux = cima - pila (P)

Si !es_vacio (C-aux) entonces

Apilar (C-aux, P-aux)

desapilar (P)

fmiembros

Devolver P-aux

fin

Ejercicio 6

espec COLAS [ELEMENTOS]

usa BOOLEANOS

Parámetro formal

genero elementos

Parámetro

genero cola

Operaciones

contar: cola → natural

ultimo: cola → elemento

Invertir: cola → cola

iguales: cola cola → bool

simétrica: cola → bool

ecuaciones

fun contar (c:cola): natural

Si es vacia? (c) entonces

Devolver 0

Sino

Devolver suc (contar (desencolar (c)))

ffun

fun último (c:cola): elemento

Si es vacia? (c) entonces

Entra (cola vacia)

Sino

aux: elemento

aux = primero (c)

mientras ! es vacia? (c) entonces

aux = desapilar (c)

mientras

Devolver aux

ffun

fun invertir (c:cola): cola

Si ! es vacia (c) entonces

Devolver encolar (primero (c), desencolar (c))

Sino

Devolver vacia

ffun

fun iguales? (c₁, c₂:cola): bool

Si es vacia? (c₁) & es vacia? (c₂) entonces

Devolver True

Sino

aux ≡ bool

aux = True

mientras ! es vacia? (c₁) & ! es vacia? (c₂) entonces

Si ! desapilar (c₁) == desapilar (c₂) entonces

aux = False

ffun

mientras

$\text{sc}^{\circ} (\text{esvacia?}(c) \vee \neg \text{esvacia?}(c)) \vee \text{aux}$

psí: $\text{aux} = \text{False}$

ffun

fun sométrica?(c; col): bool

P: pila or: bool

Si^o esvacia?(c) entonces

Devolver True

Sí no

b = True

aux: cola

aux = c

mientras ! esvacia?(aux) entonces

apilar (desencolar(aux), p)

mientras ! esvacia?(c) entonces

Si^o !(desapilar(p) = desencolar(c)) entonces

b = False

fin mientras

Devolver b

ffun

1 entragable

espec PILA [LETRAS]

Usa BOOLEANOS

Parámetro formal (síntesis de tipos) para el tipo pilas

genero Letras

f parámetro

genero letras

Operaciones

esta?: letra pila \rightarrow bool

estan_todos?: pila pila \rightarrow bool

mismas_letras?: pila_mínuscula pila_mayuscula \rightarrow bool

esta_recursivo?: letra pila \rightarrow bool

mismas_letras_recursivo?: pila_mínus pila_mayus \rightarrow bool

ecuaciones

fun esta?(l: letra, P: pila): bool

Si^o esvacia?(P) entonces

Devolver False

Sí no si Cima(P) = l entonces

Devolver True

Sí no

esta?(l, desapilar(P))

psí:

ffun

fun estan_todos(P₁: Pila, P₂: Pila): bool

si !esvacia(P₁) \wedge !esvacia(P₂) entonces

Error (Pila vacía)

Sino

b: bool

b = true

mientras !esvacia?(P₁) entonces

sc !esta?(desapilar(P₁), P₂) entonces

b = False

fsi

mientras

Devolver b

fsi

ffin

fun mismas_letras(P₁: Pila_menores, P₂: Pila_mayores): bool {mayusc()} {

si !esvacia?(P₁) \wedge !esvacia?(P₂) entonces

sino Devolver true

sc !(esvacia?(P₁) \wedge !esvacia(P₂)) \vee !(esvacia?(P₂) \wedge !esvacia(P₁)) entonces

Devolver False

Sino

sc mayusc(cima(P₁)) == cima(P₂) entonces

Devolver mismas_letras(desapilar(P₁), desapilar(P₂))

Sino

Devolver False

fsi

fsi

fsi

ffin

2 Entregable

espec PILA[LETRAS]

usa BOOLEANOS, NATURALES

parámetros formales

genero LETRAS

fp parámetros

genero PILA

d percepciones

cuantas_vocales: pila \rightarrow Natural {Recurso}

solo_vocales: pila \rightarrow pila {Recurso}

os_palabras: pila \rightarrow bool

ecuaciones

fuer cuantas_vocales (P: pila) : Naturae

Si^o es vacia? (P) entonces
Devolver 0

Sí^{no}

Si^o mayuscula(cima(P)) = 'A' V mayuscula(cima(P)) = 'E' V mayuscula(cima(P)) = 'I'
mayuscula(cima(P)) = 'O' V mayuscula(cima(P)) = 'U' entonces

Devolver

Sí^{no} suc (cuantas_vocales (desapilar(P)))

Devolver

cuantes_vocales (desapilar(P))

fun fsi

fun solo_vocales (P: pila) : pila

Si^o es vacia? (P) entonces

Devolver vacia

Sí^{no}

Si^o mayuscula(cima(P)) = 'A' V mayuscula(cima(P)) = 'E' V mayuscula(cima(P)) = 'I'
mayuscula(cima(P)) = 'O' V mayuscula(cima(P)) = 'U' entonces

Devolver

apilar (cima(P), solo_vocales(desapilar(P)))

Sí^{no}

Devolver

solo_vocales (desapilar(P))

fsi
fsi

fun es_pilobra (P: pila) : bool

Si^o es vacia? (P) entonces

Devolver False

Sí^{no}

b: bool e: letra

b=true e=desapilar(P)

mientras ! esvacia(P) entonces

:58 !(es_vocal(P) \wedge !es_vocal(e)) V !(es_vocal(P) \wedge es_vocal(e)) entonces

b=False

ps:

e = desapilar(P)

mientras

Devolver b

fun fse

Entregable

espec SISTEMA [MENSAJES]

usa BOOLEANOS, COLA, NATURAL

parámetro formal

genero Mensaje

f parámetro

género colas

Operaciones

svacio: \rightarrow cola

añadir_mensaje: Mensaje, cola (\rightarrow cola)

fpar cima_sistema: cola \rightarrow Mensaje {Recursivo}

solo_críticos: cola \rightarrow cola

llenar_col_final: cola \rightarrow cola {Recursivo}

numero_críticos: cola \rightarrow Natural

esvacio?: cola \rightarrow bool

ecuaciones

fum svacio: cola

fpar Devolver vacia

fum añadir_mensaje (m:Mensaje, (C:cola)): cola

fpar Devolver encolar (m, c)

fum cima_sistema (c:cola): Mensaje

Si esvacio? (c) entonces

Error (Sistema vacio)

Sino Si es crítico (c) entonces Devolver desencolar (c)

Sino cima_sistema (desencolar (c)) fpar

fum solo_críticos (c:cola): cola

Si estacia (c) entonces

Devolver vacia

Sino

Si es crítico (c) entonces

Devolver encolar (primero (c), solo_críticos (desencolar (c)))

Sino

Devolver solo_críticos (desencolar (c))

fpar

fpar

fun llevar_al_fondo (c:cola):cola

~~Si es vacío? (c) entonces~~
Devolver vacío
Sí no
~~Si es círculo (c) entonces~~

Sí, escíptico (.primero(c)) entonces

Devolver en color (primero(c), desencolar(c))

Sí no
Si ipsíctico (primero(c)) entonces

Devolver

fun numero_críticos (c:cola):natural

Si es vacío? (c) entonces

Devolver 0

Sí no

Si es crítico? (c) entonces

Devolver suc(numero_crítico(desencolar(c)))

Sí no

Devolver numero_crítico(desen color(c))

fs:
Pfan

fun esvacío? (c:cola):loop

Si primero(c) = NULL entonces

Devolver True

Sí no

Devolver False

fs:
Pfan

Tema -4 -

listeria dinámica \rightarrow usan pointers

3) espec LISTAR [ELEMENTO]

usa LISTA [ELEMENTO], NATURALES
Operaciones.

eliminar: elemento lista \rightarrow lista

repeticiones: elemento lista \rightarrow natural

$= =$: lista lista \rightarrow bool

equaciones

fun eliminar (e: elemento, l: lista): lista

Si^o esvacia? (l) entonces

Devolver l

Sino

Si^o prim(l) == e entonces

Devolver eliminar (resto(l))

Sino

Devolver prim(l) # elminar (resto(l))

fs:

ffun

fun repeticiones (e: elemento, l: lista): natural

Si^o esvacia? (l) entonces

Devolver 0

Sino

Si^o prim(l) == e entonces

Devolver suc (repeticiones (e, resto(l)))

Sino

Devolver repeticiones (e, resto(l))

fs:
ffun

fun $= =$ (l₁: lista, l₂: lista): bool

Si^o esvacia? (l₁) \wedge esvacia? (l₂) entonces

Devolver true

Sino esvacia? (l₁) \vee esvacia? (l₂) entonces

Devolver false

b: bool b=true

Sino

mientras ! esvacia(l₁) \wedge ! esvacia(l₂) entonces

Si^o !(prim(l₁) == prim(l₂)) entonces

b=false

fs:

resto(l₁)

resto(l₂)

mientras

Devolver b

ffun

Yespec LISTAR[ELEMENTO]

usa LISTA[ELEMENTO], NATURALES

parametros formal

Parametro \leq : elemento elemento \rightarrow bool

Operaciones

Partial minimo: lista \rightarrow elemento

Quitar: lista \rightarrow lista

Partial ordenada?: lista \rightarrow bool

excepciones

Fun minimo (l: lista): elemento

Si es vacio(l) entonces

Error (lista vacia)

Sino

e: elemento

e = prim(l)

mientras ! esvacio(l) entonces

Si prim(l) \leq e entonces

e = prim(l)

fSino

resto(l)

fmiendras

Devolver e

fsi:

ffun

Fun quitar (e: elemento, l: lista): lista

Si esvacio(l) entonces

Devolver l

Sino

Si prim(l) == e entonces

Devolver resto(l)

Sino

gostrar Devolver quitar(prim(l) #, resto(l))

fsi:

ffun

Fun ordenada? (l: lista): bool

Si esvacio(l) entonces

Error (lista vacia)

Sino

e: elemento

e = resto(l)

mientras

b: bool

b: true

mientras ! esvacio(l) entonces

Si !(prim(l) \leq e) entonces

b = false

ffun fsi Devolver b

fun insertar_en_orden(e: elemento, l: lista): lista

Si !mín_a_max(l) entonces

Error (lista no ordenada)
aux: lista

Sino

Si es vacia?(l) entonces
Devolver e:l {añadir por la izquierda}

Sino

Si mín(e, prim(l)) <= R entonces

Devolver e:l

Sino

aux = []

mientras !es vacia(l) \wedge min(prim(l), e) <= prim(l) hacer

aux = aux ++ prim(l)

l = resto(l)

mientras

Devolver aux ++ (e): l

ffun

y

espec LISTA [BOOLEAN]

usa Naturales

parametro formal

genero booleanos

parametros

genero lista

O operaciones

máximo_seguidos: lista \rightarrow natural

reducir_datos: lista \rightarrow lista

ecuaciones

fun máximo_seguidos(l:lista): natural

Si es vacia?(l) entonces

Devolver 0

Sino

b: bool

b = prim(l)

c: natural

c = 1

l = resto(l)

mientras ! esvacia(l) hacer

 se Prim(l) eq b hacer

 c = suc(c)

 Sino

 c = l

 b = prim(l)

 fisi

 l = resto(l)

fmiemtros

Devolver c

fin

fun reducir_datos(l: Pista): Pista

 Si esvacia(l) entonces

 Devolver l

 Sino aux = lista

 b: bool aux = []

 b = prim(l)

 l = resto(l) aux # b

 mientras ! esvacia(l) hacer

 Si b eq prim(l) entonces

 l = resto(l) {TTFT}

 Sino

 fisi

 b = prim(l)

 l = resto(l) aux # b

fmiemtros

Devolver aux

fin

3 espec LISTA [ELEMENTO]

usa LISTA [ELEMENTO], naturales

Parametro formal

genero lista

-<- : elemento, elemento → bool

fparametros

genero Lista2

Operaciones

fun ~~ordernar~~ ^{recursivo} ordenar_pista(l: pista): pista

 Si esvacia(l) entonces

 Devolver l

 Sino

 aux: pista

 aux = []

 mientras ! esvacia(l) entonces

 aux = Inserir_en_order(prim(l), aux)

 l = resto(l)

fmiemtros

 Devolver aux

fin

ordenar_pista: lista → lista

ordenar_pista_invertida: lista → lista

fun ordenar_lista(l: lista): lista

l = ordenar_lista(p)

l = invertir_lista(p)

Devolver l

ffun

fun invertir_lista(l: lista): lista

s: esvacua?(l) entonces

Devolver(l)

sino

{ABC}

{CBA}

Devolver Prim(l) # invertir_lista(Rest(l))

f3:

ffun

TAD PILA

Pila vacia \rightarrow crea una pila vacia

apilar \rightarrow añade un elemento en la cima de la pila

desapilar \rightarrow

desapilar: e, P \in Pila { el contenido }

la operación desapilar solo sirve si la pila no está vacía

TAD PZ LA

COLA

LISTA

LISTA +

todos los métodos básicos

en un arbol

parcializ ab : e

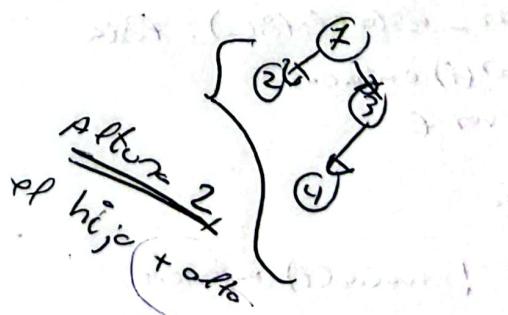
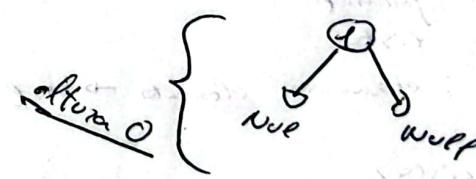
parcial Izq ab : e

parcial der ab : e

Vacio? ab : e

Vacio? ab : bool

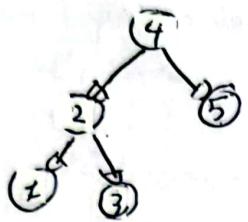
Vacio? ab



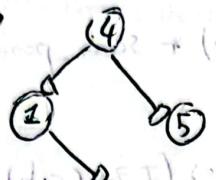
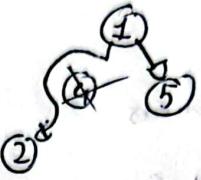
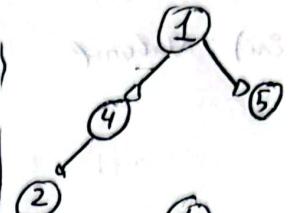
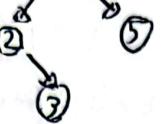
Bonnar (2, ab)

Bonne (4, ab)

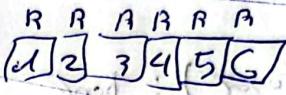
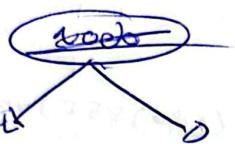
Arboles de bosques de



- $$\bullet \max = 1$$



The diagram illustrates a single neuron with a soma at the bottom right and an axon extending upwards. A bracket labeled "Profundidad" (Depth) encloses the soma and the initial segment of the axon. The number "2" is written below the soma, likely indicating the second layer of a multi-layered structure.



$$\Delta i \rightarrow a_bin$$

- - - : abin, elvisto, abin → abin

Pan	RaiZ	pre orden
	der	post orden
	Eig.	in orden or

~~209.~~
vacío?

pencil often

If m zurdo? (ab:abim): bool

Si evacuar (ab) entonces

Devolver True

Scimo

50 (es vacío? (dez(ab)) \wedge es vacío? (Izq(ab))) entonces (b) \leq a (a) \leq b

Develop true

S_i^G_{MG}

$$\int^{sc}_{\text{entances}} (\text{zundo}(\text{der}(ab)) \wedge \text{zundo}(\text{izq}(ab)) \wedge (\text{altura}(\text{izq}(ab)) > \text{altura}(\text{der}(ab))))$$

s, Devolver true

Sinc

Devolver Falso

15:

2) $\text{fun } \text{Suma_panes}(\text{ab: a_bin}): \text{Natural}$

$\text{Si } \text{esvacio?}(\text{ab})$ entonces

Devolver 0

Sino

Si $\text{par?}(\text{Raiz}(\text{ab}))$ entonces

Devolver $\text{Raiz}(\text{ab}) + \text{Suma_panes}(\text{Der}(\text{ab})) + \text{Suma_panes}(\text{Izq}(\text{ab}))$

Sino

Devolver $\text{Suma_panes}(\text{Der}(\text{ab})) + \text{Suma_panes}(\text{Izq}(\text{ab}))$

fs:

ffor

fun $\text{par?}(\text{ab: natural}): \text{bool}$

Si $(\text{N} \% 2 == 0)$ entonces

Devolver True

Sino

Devolver False

ffun

fun $\text{Imagen}(\text{ab: a_bin}): \text{a_bin}$

Si $\text{!esvacio?}(\text{ab})$ entonces

$a: a_bin$

$b: a_bin$

$a = \text{Izq}(\text{ab})$

$b = \text{DER}(\text{ab})$

Si $\text{!esvacio?}(\text{Izq}(\text{ab}))$ entonces

Si $\text{!esvacio?}(\text{Der}(\text{ab}))$ entonces Devolver $\text{a} \wedge ((\text{do}, \text{do})) \text{Imagen}$

$\text{Izq}(\text{ab}) = b$

$\text{DER}(\text{ab}) = a$

Sino

$((((\text{do}) \text{DER}(\text{ab})) = \text{Izq}(\text{ab})) \wedge ((\text{do}) \text{Izq}(\text{ab})) \text{Imagen}$

$\text{Izq}(\text{ab}) = \text{NULL}$

fs:

Sino

Si $\text{!esvacio?}(\text{Der}(\text{ab}))$ entonces

$\text{Izq}(\text{ab}) = \text{DER}(\text{ab})$

$\text{DER}(\text{ab}) = \text{NULL}$

Sino

fs:

$\text{Imagen}(\text{DER}(\text{ab}))$

$\text{Imagen}(\text{Izq}(\text{ab}))$

ffun

4
 fun pre_orden(ab: a_bim): lista
 l: lista
 $l = []$
 Si !esvacio?(ab)
 $l = \text{raiz}(ab) : l$
 $l = l ++ \text{preorden}(\text{IZQ}(ab))$
 fs: $l = l ++ \text{preorden}(\text{DER}(ab))$
 Devolver l

 fun post_orden(ab: a_bim): lista
 l: lista
 Si !esvacio?(ab) entonces
 $l = []$
 Sino
 $l = \text{post_orden}(\text{IZQ}(ab))$
 $l = l ++ \text{post_orden}(\text{Der}(ab))$
 fs: $l = l . \text{Raiz}(ab) \# l$
 Devolver l

 fun im_orden(ab: a_bim): lista
 l: lista
 Si !esvacio?(ab) entonces
 $l = []$
 Sino
 $l = \text{im_orden}(\text{IZQ}(ab))$
 $l = \text{Raiz}(ab) \# l$
 $l = l ++ \text{im_orden}(\text{Der}(ab))$
 Devolver l

d) fun esta_imorden?(ab: a_bim): boop
 Si !esvacio?(ab) entonces
 Devolver True
 Sino
 Si !esvacio?(IZQ(ab)) entonces
 Si !esvacio?(DER(ab)) entonces
 Si !(Raiz(IZQ(ab)) \leq Raiz(ab)) \wedge Raiz(ab) $<$ Raiz(DER(ab))) entonces
 Devolver Falso
 Sino
 fs: Devolver $(\text{esta_imorden?}(\text{IZQ}(ab)) \wedge \text{esta_imorden?}(\text{DER}(ab)))$
 Si !(Raiz(IZQ(ab)) \leq Raiz(ab)) entonces
 Devolver Falso

Si no
ps: Devolver esta_imondem? (IZQ(ab))

Sino
Si !esvacío? (Der(ab)) entonces

Si !(Raíz(ab) < Raíz(DER(ab))) entonces
Devolver False

Si ! Devolver esta_imondem? (DER(ab))

Sino

{Der y Izq vacíos}

ps: Devolver True

fin

3) $\text{bfm_mivel_m}(\text{ab}: \text{arbol}, n: \text{natural}): \text{Pesta}$

Si ! esvacío? (ab) entonces

• Si !(n == 0) entonces

Si ! esvacío(IZQ(ab)) entonces

Si ! esvacío(DER(ab)) entonces

Devolver mivel_m(IZQ(ab), Resto(n)) + mivel_m(DER(ab), Resto(n))

Sino

ps: Devolver mivel_m(IZQ(ab), Resto(n))

Sino

Si ! esvacío? (DER(ab)) entonces

Devolver mivel_m(DER(ab), Resto(n))

Sino

ps: Devolver []

• Sino {n == 0}

Devolver Raíz(ab)

ps:

,
Si ! {vacía ab}

Devolver []

ps:

fin

b) func niveles_entre(ab:árbol, n₁, n₂:natural): lista

Si^o esvacío?(ab) entonces

Devolver []

Sí no

Si^o (n₂ > 0) entonces

Si^o !esvacío?(izq(ab)) entonces

Si^o !esvacío?(der(ab)) entonces

Devolver niveles_entre(izq(ab), Resto(n₁), Resto(n₂)) ++
niveles_entre(der(ab), Resto(n₁), Resto(n₂))

Sí no

Devolver niveles_entre(izq(ab), Resto(n₁), Resto(n₂))

Sí no

Si^o !esvacío?(der(ab)) entonces

Devolver niveles_entre(der(ab), Resto(n₁), Resto(n₂))

Si^o Devolver []

Sí no

4/a) func num_modos(a:árbol): natural

Devolver I + num_modos_b(hijos(a))



func num_modos_b(b: bosque): natural

Si^o esvacío?(b) entonces

Devolver 0

Sí no

Devolver num_modos(primeros(b)) + num_modos_b(resto(b))

fs:

finuc

b) func num_hojas(a:árbol): Natural

Devolver Si^o vacío?(Hijos(a)) entonces I
+ num_hojas_b(Hijos(a))

finuc

Sí no Devolver num_hojas_b(Hijos(a))

func num_hojas_b(b: bosque): Natural

Si^o esvacío?(b) entonces

Devolver 0

Sí no

Devolver num_hojas(primeros(b)) + num_hojas_b(resto(b))

$\text{func } \text{max_hijos } (\alpha: \text{árbol}): \text{Natural}$

Si $\text{esvacío}(\text{bosque}(\alpha))$ entonces

Devolver 0

Sí no

$\text{num_hijos} : \text{natural}$

$\text{num_hijos} = \text{num_hijos}(\alpha)$

$\text{num_hijos_b} : \text{natural}$

$\text{num_hijos_b} = \text{max_hijos_b}(\text{bosque}(\alpha))$

Si $(\text{num_hijos} > \text{num_hijos_b})$

Devolver num_hijos

Sí no

Devolver num_hijos_b

P.S.

func

$\text{func } \text{max_hijos_b } (b: \text{bosque}): \text{Natural}$

Si $\text{esvacío}(b)$ entonces

Devolver 0

Sí no

Devolver

$\text{max}(\text{max_hijos}(\text{prim}(b)), \text{max_hijos_b}(\text{resto}(b)))$

P.S.

$\text{func } \text{frontera } (a: \text{árbol}): \text{lista}$

Si $\text{num_hijos}(a) = 0$ entonces Devolver [$\text{raiz}(a)$] si no Devolver $\text{frontera_b}(hijo(a))$

$\text{func } \text{frontera_b } (b: \text{bosque}): \text{lista}$

Si $\text{esvacío}(b)$ entonces

Devolver []

Sí no

Devolver

$\text{frontera}(\text{primo}(b)) + \text{frontera_b}(\text{resto}(b))$

P.S.

5/ TAD arbol general

-°- : elemento bosque \rightarrow árbol { dado un elemento que genere la raíz y un bosque, se crea un árbol }

[I] : \rightarrow bosque { crea una estructura árbol vacía }

-°- : árbol bosque \rightarrow bosque { crea bosques de árboles }

raiz: árbol \rightarrow elemento { devuelve la raíz del árbol, si existe }

hijos: árbol \rightarrow bosque { dada un árbol, devuelve un bosque de los hijos del árbol }

Vacio!: bosque \rightarrow bool { indica si el bosque contiene algún elemento }

Pong: bosque \rightarrow Natural { devuelve el tamaño del bosque }

num_hijos: árbol \rightarrow Natural { devuelve la cantidad de hijos }

Parcial_primer: bosque \rightarrow elemento { Devuelve el primer elemento del bosque si existe }

Parcial_nesto: bosque \rightarrow bosque { En caso de que el árbol no este vacío, borra el

Parcial_subarbol: árbol Natural \rightarrow Árbol { acceso al íesimo hijo de un árbol }

func maestro? (a:árbol): bool

Si (esvacio?(hijos(a))) entonces

Si (raiz(a) == 0) entonces

Devolver True

Si_no

Devolver False

Fin:

Si (Pong(hijos(a)) == raiz(a)) entonces

Devolver maestro_b? (hijos(a))

Devolver False

Fin:

func fin:

func maestro_b? (b:bosque): bool

Si ! esvacio?(b) entonces

9) func mayor(a: arbol): natural

Devolver max(naiz(a)), mayor(b(hijos(a)))

func mayor_b(b: bosque): natural

Si es vacio? (b) entonces

Devolver 0

Sino

Devolver mayor(mayor(primer(a)), mayor_b(resto(b)))

fs.

Func

6)

-º : elemento bosque \rightarrow arbol $\{$ crea un arbol que no tiene hijos y que va a ser la raiz y solo bosque $\}$

[] : \rightarrow bosque $\{$ crea un bosque vacio $\}$

-º - arbol \rightarrow bosque $\{$ crea bosques de arboles $\}$

raiz : arbol \rightarrow elemento $\{$ Devuelve el elemento d.

hijos : arbol \rightarrow bosque $\{$

Vacio? : bosque \rightarrow bool $\{$ un arbol jamas puede ser vacio

Pero un bosque si, por lo que si no contiene arboles este bosque dejo libera []

long : bosque \rightarrow natural $\{$ No indica el tamano del bosque

num_hijos : arbol \rightarrow natural $\{$ Muestra el numero de hijos que tiene el arbol y cuenta por los arboles que contiene ese bosque?

Primer : primer(a):

Resto : rest(b):

Subarbol : subarbol(a):