DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo práctico 1

Especificación de TADs

September 21, 2025 Algoritmos y Estructuras de Datos

JavaNation

Integrante	LU	Correo electrónico
Chioli, Lautaro	32/25	lautaro.chioli@gmail.com
Temelini, Mateo	1311/24	mateotemelini@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

http://www.exactas.uba.ar

Especificación del TAD

TAD EdR {

fotocopia ES seq⟨struct < ej: $\mathbb{Z}$, opciones: seq < $\mathbb{Z}$ >>⟩
asiento ES struct⟨*alumno:* $\mathbb{Z}$, ex: fotocopia, presente: bool⟩
aula ES seq ⟨seq < asiento >⟩

obs alumno : $\mathbb{Z}$
obs estudiantes : seq⟨alumno⟩
obs aula : aula
obs solución : seq⟨struct < ej: $\mathbb{Z}$, rta: $\mathbb{Z}$ >⟩
obs exámenes : dict < alumno, solución >
obs accesosRestantes : $\mathbb{Z}$

proc EdR(in $asientosPorFila : \mathbb{Z}$, in $resolución$: seq⟨struct⟨$ej : \mathbb{Z}, rta : \mathbb{Z}$⟩⟩, in $alumnado$: seq⟨alumno⟩, in $parciales$: fotocopia) : EdR {
  requiere {
    $alumnosVálidos(alumnado)$ ∧
    $tamañoVálido(asientosPorFila, alumnado)$ ∧
    $soluciónVálido(resolución, parciales)$ ∧
    $parcialesValidos(parciales)$
  }
  asegura {
    $|res.aula| = asientosPorFila$ ∧
    $crearAula(asientosPorFila, alumnado, parciales)$ ∧
    $exámenesInicializados(alumnado)$ ∧
    $res.estudiantes = alumnado$ ∧
    $res.solución = resoluciòn$ ∧
    $res.accesosRestantes = 0$
  }
}

pred alumnosVálidos($alumnado$: seq⟨$alumno$⟩) {
  $(\forall i : \mathbb{Z})(0 \leq i < |alumnado| \rightarrow_L (alumnado[i] \neq 0 \wedge (\forall j : \mathbb{Z})((0 \leq j < |alumnado| \wedge i \neq j) \rightarrow_L alumnado[i] \neq alumnado[j])))$
}
pred tamañoVálido($n : \mathbb{Z}, alumnado$: seq⟨$alumno$⟩) {
  $n > 0 \wedge 0 < |alumnado| \leq cantidadMáximaEstudiantes(n)$
}
aux cantidadMáximaEstudiantes($filas : \mathbb{Z}$) : $\mathbb{Z} = ((filas + 1)/2) * f$

pred soluciónValida(sol : seq⟨struct⟨$ej : \mathbb{Z}, rta : T$⟩⟩, $parciales : fotocopia$) {
  $(|sol| > 0 \wedge |sol| = |parciales|) \wedge_L (\forall t : \mathbb{Z})(0 \leq t < |sol| \rightarrow_L (\exists k : \mathbb{Z})(0 \leq k < |parciales| \wedge_L (sol[t].ej = parciales[k].ej \wedge sol[t].rta \in parciales[k].opciones)))$
}

pred parcialesVálidos($parciales : fotocopia$) {
$(\forall k : \mathbb{Z})(0 \leq k < |parciales| \rightarrow_L (|parciales[k].opciones| = 10 \ \wedge_L \ (\forall a, b : \mathbb{Z})((0 \leq a < 10 \wedge 0 \leq b < 10 \wedge a \neq b) \rightarrow_L parciales[k].opciones[a] \neq parciales[k].opciones[b]$
}
pred crearAula($n : \mathbb{Z}, alumnado : \text{seq}\langle alumno\rangle, parcial : fotocopia$) {
$existenciaEnAula(res, alumnado) \wedge unicidadEnAula(res) \wedge alumnosSeparados(res) \wedge copiaAsignada(res, alumnado, parcial)$
}
pred existenciaEnAula($salón : aula, est : \text{seq}\langle alumno\rangle, parcial : fotocopia$) {
$(\forall alumno \in est)((\exists fila, col : \mathbb{Z})((0 \leq fila < |salón| \ \wedge \ 0 \leq col < |salon[fila]|) \wedge_L (salón[fila][col].alumno = alumno \wedge salón[fila][col].ex = parcial \wedge salón[fila][col].presente = true)))$
}
pred alumnosSeparados($salón : aula$) {
$(\forall p, q : \mathbb{Z})((0 \leq p < |salón| \ \wedge \ 0 \leq q < |salón[p] - 1) \rightarrow_L (salón[p][q].alumno \neq 0 \rightarrow_L salón[p][q+1].alumno = 0$
}
pred examenesInicializados($res : EdR, est : \text{seq}\langle alumno\rangle$) {
$(\forall a : alumno)((a \in est \rightarrow_L a \in res.examenes) \ \wedge_L \ res.exámenes[a] =<>$
}

proc igualdad(in $i_1 : EdR$, in $i_2 : EdR$) : bool {
requiere { $True$ }
asegura { $res = true \iff (mismaSecuencia(i_1.estudiantes, i_2.estudiantes \ \wedge$
$mismaAula(i_1.aula, i_2.aula) \ \wedge$
$mismaSolución(i_1.solución, i_2.solución) \ \wedge$
$mismaResolución(i_1.exámenes, i_2.exámenes) \ \wedge$
$i_1.accesos\_restantes = i_2.accesos\_restantes)$ }
}
pred mismaSecuencia($a_1 : \text{seq}\langle\mathbb{Z}\rangle, a_2 : \text{seq}\langle\mathbb{Z}\rangle$) {
$|a_1| = |a_2| \wedge_L (\forall i : \mathbb{Z})(0 \leq i < |a_1| \rightarrow_L (|a_1[i]| = |a_2[i]| \wedge_L mismaDistribuciónFila(a_1[i], a_2[i])))$
}
pred mismaDistribuciónFila($f_1 : \text{seq}\langle asiento\rangle, f_2 : \text{seq}\langle asiento\rangle$) {
$(\forall j : \mathbb{Z})(0 \leq j < |f_1| \rightarrow_L (f_1[j].alumno = f_2[j].alumno \wedge mismaFotocopia(f_1[j].ex, f_2[j].ex) \wedge f_1[j].presente = f_2[j].presente))$
}
pred mismaFotocopia($f_1 : fotocopia, f_2 : fotocopia$) {
$|f_1| = |f_2| \ \wedge_L \ (\forall i : \mathbb{Z})(0 \leq i < |f_1| \rightarrow_L (f_1[i].ej = f_2[i].ej \ \wedge$
$mismaSecuencia(f_1[i].opciones, f_2[i].opciones)))$
}
pred mismaSolución($s_1 : solución, s_2 : solución$) { $|s_1| = |s_2| \wedge_L respuestasIguales(s_1, s_2) = |s_1|$ }

aux respuestasIguales($s_1 : solución, s_2 : solución$) : $\mathbb{Z} =$
$\sum_{k=0}^{|s_1|-1}(IfThenElse(ejercicioIdentico(k, s_1, s_2), 1, 0))$

```
pred ejercicioIdentico(enunciado : ℤ, sol1 : solución, sol2 : solución) {
    (∃j : ℤ)(0 ≤ j < |sol2| ∧L (sol1[enunciado].ej = sol2[j].ej ∧ sol1[enunciado].rta =
    sol2[j].rta))
}
pred mismaResolución(a1 : dict⟨alumno, solución, , ⟩a2 : dict⟨alumno, solución, )⟩ {
    (∀e : alumno)(e ∈ a1 ⟺ e ∈ a2) ∧L (∀e : alumno)(e ∈ a1 →L mismaSolución(a1[e], a2[e]))
}



proc copiarse(inout i : EdR, in a1 : alumno) {
   requiere {
       existenciaEnAula(i.aula, a1) ∧
       |i.exámenes[a1]| < |i.solución| ∧
       i = i0
   }
   asegura {
       posibilidadDeCopiarse(i, i0, a1) ∧
       i.estudiantes = i0.estudiantes ∧
       i.aula = i0.aula ∧
       i.solución = i0.solucion ∧
       i.accesos_restantes = i0.accesos_restantes
   }
}
pred posibilidadDeCopiarse(i : EdR, i0 : EdR, a1 : alumno) {
    ((∃a2 ∈ i0.estudiantes)(puedeCopiarse(i0, a1, a2)) ∧L
    agregarEjercicio(i0.exámenes[a1], i0.exámenes[a2], i.exámenes[a1])) ∨L
    (¬∃a2 ∈ i0.estudiantes)(puedeCopiarse(i0, a1, a2)) ∧L
    mismaResolución(i.examenes, i0.examenes)
}
pred puedeCopiarse(i : EdR, a1 : alumno, a2 : alumno) {
    (a2 ≠ a1) ∧L (∃p1, p2 : ⟨fila : ℤ, col : ℤ⟩)((sacarPosición(i.aula, a1, p1) ∧ sacarPosición(i.aula, a2, p
    sonCercanos(p1, p2)) ∧L existeEjerciciosDisponibles(i.examenes[a1], i.examenes[a2])
}
pred sonCercanos(ubi1 : ⟨fila : ℤ, col : ℤ⟩, ubi2 : ⟨fila : ℤ, col : ℤ⟩) {
    (u2.fila = u1.fila ∧ |u2.col − u1.col| ≤ 2) ∨
    (u2.fila = u1.fila − 1 ∧ |u2.col − u1.col| ≤ 2)
}
pred sacarPosición(salón : aula, a : alumno, pos : ⟨fila : ℤ, col : ℤ⟩) {
    (∃f : ℤ)(0 ≤ f < |salón| ∧L (∃c : ℤ)(0 ≤ c < |salón[f]| ∧L salón[f][c].alumno =
    a ∧L (pos.fila = f ∧ pos.col = c)))
}
pred existeEjercicioDisponible(p1 : solución, p2 : solución) {
    (∃punto : ⟨ej : ℤ, rta : ℤ⟩)(punto ∈ p2 ∧ ¬(∃t ∈ p1)(t.ej = punto.ej))
}
pred agregarEjercicio(p1 : solución, p2 : solución, res : solución) {
    |res| = |p1| + 1 ∧ (∃punto : ⟨ej : ℤ, rta : ℤ⟩)(punto ∈ p2 ∧ ¬(∃t : ⟨ej : ℤ, rta :
    ℤ⟩)(t ∈ p1 ∧ t.ej = punto.ej) ∧L res = p1 + + < punto >
}
```

```
proc consultarDarkWeb(i : EdR, in examenDark : solución, in entradas : ℤ) {
   requiere {
        |examenDark| = |i.solución| ∧
        entradas ≤ i.accesos_restantes ∧
        i = i0
   }
   asegura {
        (∃consultantes : seq⟨alumno⟩)(|consultantes| ≤ entradas ∧L
        examenCopiado(i, i0, examenDark, consultantes)) ∧
        i.estudiantes = i0.estudiantes ∧ i.aula = i0.aula ∧ i.solución = i0.solución ∧
        i.accesos_restantes = i0.accesos_restantes − entradas
   }
}

pred examenCopiado(i : EdR, i0 : EdR, solDark : solución, est : seq⟨alumno⟩) {
    (∀a : alumno)((a ∈ est ∧ a ∈ i0.estudiantes) →L i.examenes[a] = solDark) ∧ (∀b :
    alumno)((b ∉ est ∧ b ∈ i0.estudiantes) →L i.examenes[b] = i0.examenes[b])
}

proc resolver(i : EdR, in a : alumno, in pasos : seq⟨solución⟩) : seq⟨solución⟩ {
   requiere { existenciaEnAula(i.aula, a) ∧ i.examenes[a] =<> ∧ i = i0 ∧
   pasosValidos(pasos, i.solucion) }
   asegura {
        |res| = |pasos| + 1 ∧L (∃ejAux : struct < ej : ℤ, rta : ℤ >)
        (actualizarExamenes(i, i0, a, pasos, res, ejAux)) ∧
        i.estudiantes = i0.estudiantes ∧ i.aula = i0.aula ∧ i.solución = i0.solución ∧
        i.accesos_restantes = i0.accesos_restantes
   }
}

pred actualizarExámenes(i : EdR, i0 : EdR, a : alumno, pasos : seq⟨solución⟩, res :
seq⟨solución⟩, ejAux : ⟨ej : ℤ, rta : ℤ⟩) {
    ejercicioValido(i.solución, ejAux, pasos) ∧ secuenciaFinal(pasos, res, ejAux) ∧
    verificarExámenes(i.exámenes, i0.exámenes, a, res)
}
pred pasosVálidos(pasos : seq⟨solución⟩, sol : solución) {
    (1 ≤ |pasos| ≤ |sol| + 1 ∧L pasos[0] =<>) ∧ (∀k : ℤ)(0 ≤ k < |pasos| − 1 →L (∀j :
    ℤ)(0 ≤ j < |pasos[k]| → pasos[k][j] ∈ pasos[k + 1]))
}
pred ejercicioVálido(s : solución, resp : ⟨ej : ℤ, rta : ℤ⟩, procesos : seq⟨solución⟩) {
    (∃k : ℤ)(0 ≤ k < |s| ∧ s[k].ej = resp.ej) ∧ (¬∃j : ℤ)(0 ≤ j < |proceso[|proceso| −
    1]| ∧ proceso[|proceso| − 1][j].ej = resp.ej))(
}
```

pred secuenciaFinal($entrada$: seq$\langle$*solución*$\rangle$, $salida$: seq$\langle$*solución*$\rangle$, ad : $\langle ej : \mathbb{Z}, rta : \mathbb{Z}\rangle$) {

$(\forall i : \mathbb{Z})(0 \leq i < |entrada| \rightarrow_L salida[i] = entrada[i]) \ \wedge \ salida[|salida| - 1] = salida[|entrada| - 1] + + < ad.ej, ad.rta >$

}

pred verificarExámenes(exs : dict$\langle$*alumno*, *solución*, , $\rangle exs_0$: dict$\langle$*alumno*, *solución*, , $\rangle a$: alumno, $salida$: seq$\langle$*solución*$\rangle$) {

$(\forall c \in exs_0)(c \neq a \rightarrow_L exs[c] = exs_0[c]) \ \wedge \ exs[a] = salida[|salida| - 1]$

}

proc entregar(inout i : EdR, in a : alumno) {

requiere { $existenciaEnAula(i.aula, a) \ \wedge \ i = i0$ }

asegura { $i.estudiantes = i0.estudiantes \wedge i.solucion = i0.solucion \wedge i.examenes = i0.examenes \ \wedge \ i.accesos\_restantes = i0.accesos\_restantes$ }

}

pred quitarEstudiante($salón : aula, a : alumno$) {

$|res| = |salón| \wedge_L (alumnoRemovido(salón, a, res) \wedge restoPresente(salón, a, res))$

}

pred alumnoRemovido($salón : aula, a : alumno, res : aula$) {

$(\forall fila : \mathbb{Z})(0 \leq fila < |salón| \rightarrow_L (\forall col\mathbb{Z})(0 \leq col < |salón[fila]| \rightarrow_L (salón[fila][col].alumno = a \rightarrow_L (res[fila][col].alumno = a \ \wedge \ res[fila][col].presente = false))))$

}

pred restoPresente($salón : aula, a : alumno, res : aula$) {

$(\forall fila : \mathbb{Z})(0 \leq fila < |salón| \rightarrow_L (\forall col\mathbb{Z})(0 \leq col < |salón[fila]| \rightarrow_L (salón[fila][col].alumno \neq a \rightarrow_L (res[fila][col] \ \wedge \ salón[fila][col]))))$

}

proc chequearCopias(in i : EdR) : estudiantes {

requiere { $aulaVacia(i.aula, i.estudiantes$ }

asegura { $(\forall est \in i.estudiantes)(est \in res \leftrightarrow examenRepetido(est, i) \vee esSospechoso(est, i))$ }

}

pred examenRepetido($a : alumno, i : EdR$) {

$(\exists alumnado$: seq$\langle alumno\rangle)((|alumnado| \geq |i.estudiantes|/4 \ \wedge \ a \in alumnado) \ \wedge_L (\forall b \in alumnado)(mismaSolución(i.exámenes[a], i.exámenes[b])))$

}

pred aulaVacia($salón : aula, i : EdR$) {

$(\forall a : alumno)(a \in i.estudiantes \rightarrow_L \neg existenciaEnAula(salón, a))$

}

pred esSospechoso($a : alumno, i : EdR$) {

$(\exists e \in i.estudiantes)(a \neq e \wedge_L (sonCercanos(sacarPosición(i.aula, a), sacarPosición(i.aula, e)) \wedge respuestasIguales(i.exámenes[a], i.exámenes[e]) \geq 0,6 * |i.solución|))$

}

```
    proc corregir(in i : EdR) : seq⟨struct < a : alumno, n : ℝ >⟩ {
        requiere { aulaVacia(i.aula, i.estudiantes) }
        asegura { (∀alumno ∈ i.estudiante)(esSospechoso(alumno, i) →_L ¬recibiraNota(alumno, i, res))
        }
        asegura { (∀alumno ∈ i.estudiante)(¬esSospechoso(alumno, i) →_L recibiraNota(alumno, i, res))
        }
    }
    pred recibiraNota(a : alumno, i : Edr, corregidos : seq⟨struct < a: alumno, n: ℝ >⟩)
    {
        (∃k : ℤ, n : ℝ)(0 ≤ k < |corregidos| ∧_L corregidos[k].a = a ∧
        verificarNota(a, corregidos, i.solución, i.exámenes[a]))
    }
    pred verificarNota(a : alumno, res : seq⟨struct < a: alumno, n: ℝ >, s : solución, p :
    solución⟩) {
        (∃j : ℤ)(0 ≤ j < |res| ∧_L res[j].a = a ∧ respuestasIguales(s, p) / |s|)
    }
}
```