



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo práctico

Especificación de TADs

September 11, 2025

Algoritmos y Estructuras de Datos

BobElConstructorPorCopia

Integrante	LU	Correo electrónico
Choque, Leandro	252/25	leandroch2002@gmail.com
Musi, Santino	965/24	santinomusi1@gmail.com
Rojas, Damian	209/25	dam.rojas1@gmail.com
Martell, Juan Bautista	622/25	Juanbamartell@hotmail.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>

1 Supongo que acá iría una descripción

Breve descripción.

Luego veremos bien el formato, esto de momento es para tener un esqueleto.

2 Especificacion

```
TAD EdR {
  obs aula : Aula
  obs solucion : Paso
  obs entregas : seq⟨Alumno⟩

  proc EdR(in dimensionAula : ℤ, in s : Paso, in cantEstudiantes : ℤ) : EdR {
    requiere {
      (dimensionAula > 0) ∧L
      rtaValida(s) ∧L
      cantValidaEstudiantes(dimensionAula, cantEstudiantes)
    }
    asegura {
      (|res.aula| = dimensionAula) ∧L
      aulaCuadrada(res.aula) ∧L
      noHayAlumnosJuntos(res.aula) ∧L
      (cuantosEstudiantes(res.aula) = cantEstudiantes) ∧L
      examenesSinResponder(res.aula) ∧L
      (res.solucion = s) ∧L
      (res.entregas = ⟨⟩)
    }
  }
}

pred rtaValida(s : Paso) {
  (∀i : ℤ) (0 ≤ i < |s| →L s[i] ∈ conj⟨"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"⟩)
}

pred cantValidaEstudiantes(a : Aula, e : ℤ) {
  (e ≤ ifThenElseFi(mod(|a|, 2) == 0,  $\frac{|a|^2}{2}$ ,  $\frac{|a|+1}{2} * |a|$ ))
}

pred aulaCuadrada(a : Aula) {
  (∀i : ℤ) (0 ≤ i < |a| →L |a[i]| = |a|)
}

pred noHayAlumnosJuntos(a : Aula) {
  (∀i : ℤ) (0 ≤ i < |a| →L
    (∀j : ℤ) (0 ≤ j < |a[i]| - 1 →L (a[i][j] ≠ ⟨⟩ → a[i][j+1] = ⟨⟩)))
}

aux cuantosEstudiantes(a : Aula) : ℤ =  $\sum_{i=0}^{|a|-1} \sum_{j=0}^{|a[i]|-1} \text{ifThenElseFi}(a[i][j] \neq \langle \rangle, 1, 0)$ 
pred examenesSinResponder(a : Aula) {
  (∀i : ℤ) (0 ≤ i < |a| →L (∀j : ℤ) (0 ≤ j < |a[i]| →L examenSinResponder(a[i][j].examen)))
}

pred examenSinResponder(e : Examen) {
  (|e| = 1 ∧ (∀i : ℤ) (0 ≤ i < |e| → e[0][i] = ""))
}

proc igualdad(in edr1, edr2 : EdR, ) : Bool {
```

```

    requiere { True }
    asegura {
      (res = True)  $\leftrightarrow$ 
      (edr1.aula = edr2.aula)  $\wedge$ 
      (edr1.solucion = edr2.solucion)  $\wedge$ 
      (edr1.entregas = edr2.entregas)
    }
  }
}

proc copiarse(in alumno : Alumno, inout edr : EdR) : {
  requiere {
    (perteneceAlumno(alumno, edr.aula)  $\wedge_L$ 
    edr = edr0  $\wedge_L$ 
    alumno.cordenadas.x = x0  $\wedge_L$ 
    alumno.cordenadas.y = y0  $\wedge_L$ 
  }
  asegura {
    ( $\exists i, j : \mathbb{Z}$ ) (cordenadasValidas(i, j, edr.Aula)  $\wedge_L$ 
    adyacente(< i, j >, alumno.cordenadas)  $\wedge_L$ 
    ( $\exists k : \mathbb{Z}$ ) (alumno.examen[|alumno.examen| - 1][k] = ""  $\wedge$ 
    edr.Aula[i][j].examen[|examen|][k]  $\neq$  ""  $\wedge$ 
    edr.Aula[i][j].examen[|examen|][k] = edr0.Aula[x0][y0].examen[|alumno.examen|][k])  $\wedge_L$ 

    ( $\forall p : \mathbb{Z}$ ) (( $\forall q : \mathbb{Z}$ ) (cordenadasValidas(p, q, edr0.Aula)  $\wedge_L$ 
    (x0  $\neq$  p  $\wedge$  y0  $\neq$  q)  $\rightarrow_L$  edr.Aula[p][q] = edr0.Aula[p][q]))  $\wedge_L$ 
    edr0.solucion = edr.solucion  $\wedge_L$ 
    edr0.entregas = edr.entregas
  }
}

}

proc consultarDarkWeb(in s : Paso, in posiblesAccesos :  $\mathbb{Z}$ , inout edr : EdR) {
  requiere {
    rtaValida(s)  $\wedge_L$ 
    posiblesAccesos  $\geq$  0  $\wedge_L$ 
  }
  asegura { res }
}

}

proc resolver(in alumno : Coordinada, inout edr : EdR) : Paso {
  requiere {
    (existeAlumno(alumno))  $\wedge_L$ 
    ( $\exists i : \mathbb{Z}$ ) (0  $\leq$  i < |alumno.examen[0]|  $\wedge_L$  alumno.examen[|alumno.examen|][k] ==
    "")
  }
  asegura { res }
}

}

proc entregar(in alumno : Coordinada) : EdR {
  requiere { True }
  asegura { res }
}

}

proc chequearCopias(in alumnos : seq < Alumno >) : seq < Alumno > {
  requiere { True }

```

```

    asegura { res }
  }
proc corregir(inout edr, EdR) : seq << Alumno, Nota >> {
  requiere { True }
  asegura { res }
}
}

```