Practica 3 Algoritmos y Estructuras de Datos

Aitor

12 de abril de 2025

Ejercicio 1. Especificar en forma completa el TAD NumeroRacional que incluya las operaciones aritméticas básicas (suma, resta, división, multiplicación) y una operación igual que dados dos números racionales devuelva verdadero si son iguales.
Solución:

TAD *NumeroRacional* {
obs numerador: $\mathbb{Z}$obs denominador: $\mathbb{Z}$

proc nuevoNumeroRacional (in n:$\mathbb{Z}$, in d:$\mathbb{Z}$) : numeroRacional
requiere $\{d \neq 0\}$
asegura $\{res.numerador = n \wedge res.denominador = d\}$

proc suma (inout a: numeroRacional, in b: numeroRacional)
requiere $\{a = a0\}$
asegura $\{a.numerador = a0.numerador * b.denominador + b.numerador * a0.denominador \wedge$
$a.denominador = a0.denominador * b.denominador\}$

proc resta (inout a: numeroRacional, in b: numeroRacional)
requiere $\{a = a0\}$
asegura $\{a.numerador = a0.numerador * b.denominador - b.numerador * a0.denominador \wedge$
$a.denominador = a0.denominador * b.denominador\}$

proc multiplicación (inout a: numeroRacional, in b: numeroRacional)
requiere $\{a = a0\}$
asegura $\{a.numerador = a0.numerador * b.numerador \wedge a.denominador = a0.denominador * b.denominador\}$

proc división (inout a: numeroRacional, in b: numeroRacional)
requiere $\{a = a0\}$
requiere $\{b.numerador \neq 0\}$
asegura $\{a.numerador = a0.numerador * b.denominador \wedge a.denominador = a0.denominador * b.numerador\}$
}

Ejercicio 2. Especifique mediante TADs los siguientes elementos geométricos:

a) Punto2D, que representa un punto en el plano. Debe contener las siguientes operaciones:
   a) nuevoPunto: que crea un punto a partir de sus coordenadas x e y.
   b) mover : que mueve el punto una determinada distancia sobre los ejes x e y.
   c) distancia: que devuelve la distancia entre dos puntos.
   d) distanciaAlOrigen: que devuelve la distancia del punto (0, 0).

Solución:

TAD *Punto2D* {
obs x: $\mathbb{R}$
obs y: $\mathbb{R}$

proc nuevoPunto (in x:$\mathbb{R}$, in y:$\mathbb{R}$) : punto2D
requiere $\{True\}$
asegura $\{res.x = x \wedge res.y = y\}$

`proc mover` (inout p: punto2D, in dx:$\mathbb{R}$, in dy:$\mathbb{R}$)

`requiere` $\{p = p0\}$

`asegura` $\{p.x = p0.x + dx \wedge p.y = p0.y + dy\}$

`aux distanciaEntrePuntos` (in p1: punto2D, in p2: punto2D) : $\mathbb{R} = \sqrt{(p1.x - p2.x)^2 + (p1.y - p2.y)^2}$;

`proc distancia` (in p1: punto2D, in p2: punto2D) : $\mathbb{R}$

`requiere` $\{p1 = p0\}$

`asegura` $\{res = distanciaEntrePuntos(p0, p2)\}$

`proc distanciaAlOrigen` (in p: punto2D) : $\mathbb{R}$

`requiere` $\{p = p0\}$

`asegura` $\{res = distanciaEntrePuntos(p0, nuevoPunto(0, 0))\}$

}

b) Rectángulo2D, que representa un rectángulo en el plano. Debe contener las siguientes operaciones:

a) nuevoRectángulo: que crea un rectángulo (decida usted cuáles deberían ser los parámetros).

b) mover : que mueve el rectángulo una determinada distancia en los ejes x e y.

c) escalar : que escala el rectángulo en un determinado factor. Al escalar un rectángulo un punto del mismo debe quedar fijo. En este caso el punto fijo puede ser el centro del rectángulo o uno de sus vértices.

d) estáContenido: que dados dos rectángulos, indique si uno está contenido en el otro.

Solución:

`TAD` *Rectangulo2D* {

obs x1: $\mathbb{R}$
obs y1: $\mathbb{R}$
obs x2: $\mathbb{R}$
obs y2: $\mathbb{R}$

`proc nuevoRectangulo` (in x1:$\mathbb{R}$, in y1:$\mathbb{R}$, in x2:$\mathbb{R}$, in y2:$\mathbb{R}$) : rectangulo2D

`requiere` $\{True\}$

`asegura` $\{res.x1 = x1 \wedge res.y1 = y1 \wedge res.x2 = x2 \wedge res.y2 = y2\}$

`proc mover` (inout r: rectangulo2D, in dx:$\mathbb{R}$, in dy:$\mathbb{R}$)

`requiere` $\{r = r0\}$

`asegura` $\{r.x1 = r0.x1 + dx \wedge r.y1 = r0.y1 + dy \wedge r.x2 = r0.x2 + dx \wedge r.y2 = r0.y2 + dy\}$

`aux distancia` (in x: $\mathbb{R}$, in y: $\mathbb{R}$) : $\mathbb{R} = \sqrt{(x - y)^2}$;

`proc escalar` (inout r: rectangulo2D, in factor:$\mathbb{R}$)

`requiere` $\{r = r0\}$

`asegura` $\{r.x1 = r0.x1 \wedge r.y1 = r0.y1 \wedge$
$r.x2 = r0.x1 + distancia(x1, x2) * factor \wedge$
$r.y2 = r0.y1 + distancia(y1, y2) * factor\}$

`proc estaContenido` (in r1: rectangulo2D, in r2: rectangulo2D) : Bool

`requiere` $\{True\}$

`asegura` $\{res = (r1.x1 \leq r2.x1 \wedge r1.y1 \leq r2.y1 \wedge r1.x2 \geq r2.x2 \wedge r1.y2 \geq r2.y2) \vee$
$(r2.x1 \leq r1.x1 \wedge r2.y1 \leq r1.y1 \wedge r2.x2 \geq r1.x2 \wedge r2.y2 \geq r1.y2)\}$

}

Ejercicio 3.

a) Especifique el TAD Cola⟨T⟩ con las siguientes operaciones:

a) nuevaCola: que crea una cola vacía
b) estáVacía: que devuelve true si la cola no contiene elementos
c) encolar: que agrega un elemento al final de la cola
d) desencolar: que elimina el primer elemento de la cola y lo devuelve

Solución:

TAD *Cola⟨T⟩* {

obs elementos: seq⟨T⟩

proc nuevaCola () : Cola⟨T⟩

requiere $\{True\}$

asegura $\{|res.elementos| = 0\}$

proc estáVacía (in c: Cola⟨T⟩) : Bool

requiere $\{True\}$

asegura $\{res = (|c.elementos| = 0)\}$

proc encolar (inout c: Cola⟨T⟩, in e: T)

requiere $\{c = c0\}$

asegura $\{|c.elementos = c0.elementos + 1|\}$

asegura $\{(\forall i : \mathbb{Z})\ ((0 \leq i < |c0.elementos|) \longrightarrow_L c.elementos[i] = c0.elementos[i] \wedge c.elementos[|c0.elementos|] = e)\}$

proc desencolar (inout c: Cola⟨T⟩) : T

requiere $\{c = c0\}$

requiere $\{|c0.elementos| > 0\}$

asegura $\{res = c0.elementos[0]\}$

asegura $\{|c.elementos| = |c0.elementos| - 1\}$

asegura $\{(\forall i : \mathbb{Z})\ ((0 \leq i < |c.elementos|) \longrightarrow_L c.elementos[i] = c0.elementos[i + 1])\}$

}

b) Especifique el TAD Pila⟨T⟩ con las siguientes operaciones:

a) nuevaPila: que crea una pila vacía
b) estáVacia: que devuelve true si la pila no contiene elementos
c) apilar: que agrega un elemento al tope de la pila
d) desapilar: que elimina el elemento del tope de la pila y lo devuelve

Solución:

TAD *Pila⟨T⟩* {

obs elementos: seq⟨T⟩

proc nuevaPila () : Pila⟨T⟩

requiere $\{True\}$

asegura $\{|res.elementos| = 0\}$

proc estáVacia (in p: Pila⟨T⟩) : Bool

requiere $\{True\}$

asegura $\{res = (|p.elementos| = 0)\}$

proc apilar (inout p: Pila⟨T⟩, in e: T)

requiere $\{p = p0\}$

asegura $\{|p.elementos| = |p0.elementos| + 1|\}$

asegura $\{(\forall i : \mathbb{Z})\ ((0 \leq i < |p.elementos|) \longrightarrow_L p.elementos[i] = p0.elementos[i] \wedge p.elementos[|p0.elementos|] = e)\}$

proc desapilar (inout p: Pila⟨T⟩) : T

requiere $\{p = p0\}$

requiere $\{|p0.elementos| > 0\}$

asegura $\{res = p0.elementos[|p0.elementos| - 1]\}$

asegura $\{|p.elementos| = |p0.elementos| - 1|\}$

asegura $\{(\forall i : \mathbb{Z})\ ((0 < i < |p.elementos|) \longrightarrow_L p.elementos[i] = p0.elementos[i - 1])\}$

}

c) Especifique el TAD DobleCola⟨T ⟩, en el que los elementos pueden insertarse al principio o al final y se eliminan por el medio. Debe contener las operaciones nuevaDobleCola, estáVacía, encolarAdelante, encolarAtrás y desencolar.

TAD *DobleCola⟨T⟩* {

obs elementos: seq⟨T⟩

proc nuevaDobleCola () : DobleCola⟨T⟩

requiere $\{True\}$

asegura $\{|res.elementos| = 0\}$

proc estáVacía (in dc: DobleCola⟨T⟩) : Bool

requiere $\{True\}$

asegura $\{res = (|dc.elementos| = 0)\}$

proc encolarAdelante (inout dc: DobleCola⟨T⟩, in e: T)

requiere $\{dc = dc0\}$

asegura $\{|dc.elementos| = |dc0.elementos| + 1|\}$

asegura $\{(\forall i : \mathbb{Z})\ ((0 < i < |dc.elementos|) \longrightarrow_L dc.elementos[i] = dc0.elementos[i - 1] \wedge dc.elementos[0] = e)\}$

proc encolarAtras (inout dc: DobleCola⟨T⟩, in e: T)

requiere $\{dc = dc0\}$

asegura $\{|dc.elementos| = |dc0.elementos| + 1|\}$

asegura $\{(\forall i : \mathbb{Z})\ ((0 < i < |dc.elementos|) \longrightarrow_L dc.elementos[i] = dc0.elementos[i] \wedge dc.elementos[|dc0.elementos|] = e)\}$

proc desencolar (inout dc: DobleCola⟨T⟩) : T

requiere $\{dc = dc0\}$

requiere $\{|dc0.elementos| > 0\}$

asegura $\{res = dc0.elementos[|dc0.elementos|/2]\}$

asegura $\{|dc.elementos| = |dc0.elementos| - 1|\}$

asegura $\{(\forall i : \mathbb{Z})\ ((0 \leq i < |dc0.elementos|/2) \longrightarrow_L dc.elementos[i] = dc0.elementos[i] \wedge (|dc0.elementos|/2 < i < |dc0.elementos|) \longrightarrow_L dc.elementos[i - 1] = dc0.elementos[i])\}$

}

Ejercicio 4.

a) Especifique el TAD Diccionario⟨K, V⟩ con las siguientes operaciones:

a) nuevoDiccionario: que crea un diccionario vacío
b) definir : que agrega un par clave-valor al diccionario
c) obtener : que devuelve el valor asociado a una clave
d) está: que devuelve true si la clave está en el diccionario
e) borrar : que elimina una clave del diccionario

`TAD` *Diccionario⟨K, V⟩* {

obs clavesValores: seq⟨struct⟨key: K, val: V⟩⟩

`proc nuevoDiccionario` () : Diccionario⟨K, V⟩

`requiere` $\{True\}$

`asegura` $\{|res.clavesValores| = 0\}$

`proc definir` (inout d: Diccionario⟨K, V⟩, in k: K, in v: V)

`requiere` $\{d = d0\}$

`asegura` $\{|d.clavesValores| = |d0.clavesValores| + 1|\}$

`asegura` $\{(\forall i : \mathbb{Z})\ ((0 < i < |d.clavesValores|) \longrightarrow_L d.clavesValores[i] = d0.clavesValores[i] \wedge (d.clavesValores[|d0.clavesValores|].key = k \wedge d.clavesValores[|d0.clavesValores|].val = v))\}$

`proc obtener` (in d: Diccionario⟨K, V⟩, in k: K) : V

`requiere` $\{|d.clavesValores| > 0\}$

`requiere` $\{esta(d, k)\}$

`asegura` $\{(\exists i : \mathbb{Z})\ ((0 \leq i < |d.clavesValores|) \wedge d.clavesValores[i].key = k \wedge res = d.clavesValores[i].val)\}$

`proc esta` (in d: Diccionario⟨K, V⟩, in k: K) : Bool

`requiere` $\{True\}$

`asegura` $\{res = (\exists i : \mathbb{Z})\ ((0 \leq i < |d.clavesValores|) \wedge d.clavesValores[i].key = k)\}$

`proc borrar` (inout d: Diccionario⟨K, V⟩, in k: K)

`requiere` $\{d = d0\}$

`requiere` $\{esta(d0, k)\}$

`asegura` $\{|d.clavesValores| = |d0.clavesValores| - 1|\}$

`asegura` $\{(\forall i : \mathbb{Z})\ ((0 \leq i < |d.clavesValores|) \longrightarrow_L (d.clavesValores[i].key \neq k \longrightarrow_L d.clavesValores[i] = d0.clavesValores[i]))\}$

}

b) Especifique el TAD DiccionarioConHistoria⟨K, V ⟩. El mismo permite consultar, para cada clave, todos los valores que se asociaron con la misma a lo largo del tiempo (en orden). Se debe poder hacer dicha consulta aún si la clave fue borrada.

Ejercicio 5. Especifique los TADs indicados a continuación pero utilizando los observadores propuestos:

a) Diccionario⟨K, V ⟩ observado con conjunto (de tuplas)

b) Pila⟨T ⟩ observado con diccionarios

c) Punto observado con coordenadas polares

Ejercicio 6. Especificar TADs para las siguientes estructuras:

a) Multiconjunto⟨T ⟩ También conocido como multiset o bag. Es igual a un conjunto pero con duplicados: cada elemento puede agregarse múltiples veces. Tiene las mismas operaciones que el TAD Conjunto, más una operación que indica la multiplicidad de un elemento (la cantidad de veces que ese elemento se encuentra en la estructura). Nótese que si un elemento es eliminado del multiconjunto, se reduce en 1 la multiplicidad. Ejemplo:

b) Multidict⟨K, V ⟩ Misma idea pero para diccionarios: Cada clave puede estar asociada con múltiples valores. Los valores se definen de a uno (indicando una clave y un valor), pero la operación obtener debe devolver todos los valores asociados a una determinada clave. *Nota:* En este ejercicio deberá tomar algunas decisiones. ¿Se pueden asociar múltiples veces un mismo valor con una clave? ¿Qué pasa en ese caso? Qué parámetros tiene y cómo se comporta la operación borrar? Imagine un caso de uso para esta estructura y utilice su sentido común para tomar estas decisiones.

Ejercicio 7. Especifique el TAD Contadores que, dada una lista de eventos, permite contar la cantidad de veces que se produjo cada uno de ellos. La lista de eventos es fija. El TAD debe tener una operación para incrementar el contador asociado a un evento y una operación para conocer el valor actual del contador para un evento.

- Modifique el TAD para que sea posible preguntar el valor del contador en un determinado momento del pasado. Si necesita conocer la fecha y hora actual, puede pasarla como parámetro a los procedimientos. Asuma que las fechas son números enteros (por ejemplo, la cantidad de segundos desde el 1 de enero de 1970).