



Trabajo practico 1

Especificacion y WP

15 de septiembre de 2024

Algoritmos y Estructuras de Datos

X-force

Integrante	LU	Correo electrónico
Krivososoff, Thiago	310/24	thiagokribas@gmail.com
Pellitero, Agustin	185/24	agustinignaciopelli@gmail.com
Miguel, Facundo	702/24	facumiguel4025@gmail.com
Montenegro, Ulises	477/24	ulinicolasmonte@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>

1. Punto 1

1.1. Ejercicio 1

```
proc grandesCiudades (in ciudades : seq<Ciudad>) : seq<Ciudad>
  requiere {noRepetidos(ciudades) ∧ noHabitantesNegativos(ciudades)}
  asegura {|res| ≤ |ciudades|}
  asegura {(∀elem : Ciudad) ((elem.habitantes > 50000 ∧ elem ∈ ciudades) ↔ elem ∈ res)}

pred noRepetidos (in ciudades: seq<Ciudad>) { (∀i, j : Z) (0 ≤ i, j < |ciudades|) →L (i ≠ j → ciudades[i].nombre ≠ ciudades[j].nombre) }
pred noHabitantesNegativos (in ciudades: seq<Ciudad>) { (∀i : Z) (0 ≤ i < |ciudades|) →L ciudades[i].habitantes >= 0 }
```

1.2. Ejercicio 2

```
proc sumaDeHabitantes (in menoresDeCiudades: seq<Ciudad>, in mayoresDeCiudades: seq<Ciudad>) : seq<Ciudad>
  requiere {noRepetidos(menoresDeCiudades) ∧ noRepetidos(mayoresDeCiudades)}
  requiere {noHabitantesNegativos(menoresDeCiudades) ∧ noHabitantesNegativos(mayoresDeCiudades)}
  requiere {|menoresDeCiudades| = |mayoresDeCiudades| ∧L
  mismosElementos(menoresDeCiudades, mayoresDeCiudades)}
  asegura {|res| = |mayoresDeCiudades|}
  asegura {(∀elem : Ciudad) ((elem ∈ res) ↔
  (∀i : Z) (0 ≤ i < |res|) →L (∃j : Z) (0 ≤ j < |mayoresDeCiudades| ∧
  mayoresDeCiudades[j].nombre = menoresDeCiudades[i].nombre →
  (res[i].nombre = mayoresDeCiudades[j].nombre ∧
  res[i].habitantes = mayoresDeCiudades[j].habitantes + menoresDeCiudades[j].habitantes))}

pred mismosElementos (in s1: seq<Ciudad>, in s2: seq<Ciudad>) {
  (∀i : Z) (0 ≤ i < |s1|) →L (∃j : Z) (0 ≤ j < |s2| ∧ s1[i].nombre = s2[j].nombre) }
```

1.3. Ejercicio 3

```
proc hayCamino (in distancia: seq<seq<Z>>, in desde: Z, in hasta: Z) : Bool
  requiere {esCuadrada(distancia) ∧L filaIgualColumna(distancia)}
  requiere {0 ≤ desde < |distancia|}
  requiere {0 ≤ hasta < |distancia|}
  requiere {matrizTodosPositivos(distancia)}
  asegura {res = True ↔ (∃sec : seq<Z>) (|sec| > 1) ∧L secuenciaEsCamino(distancia, sec, desde, hasta)}

pred secuenciaEsCamino (in distancia: seq<seq<Z>>, in sec: seq<Z>, in desde: Z, in hasta: Z) { sec[0] = desde ∧ sec[|sec|-1] = hasta ∧ (∀i : Z) (0 ≤ i < |sec|) →L
0 ≤ sec[i] < |distancia| ∧L todosConexionAnterior(sec, distancia) }
pred todosConexionAnterior (in sec: seq<Z>, in mat: seq<seq<Z>>) { (∀i : Z) (1 ≤ i < |sec|) →L mat[sec[i]][sec[i-1]] ≠ 0 }
pred esCuadrada (in mat: seq<seq<Z>>) { (∀i : Z) (0 ≤ i < |mat|) →L |mat| = |mat[i]| }
pred filaIgualColumna (in mat: seq<seq<Z>>) { (∀i, j : Z) (0 ≤ i, j < |mat|) →L mat[i][j] = mat[j][i] }
pred matrizTodosPositivos (in mat: seq<seq<Z>>) { (∀i, j : Z) (0 ≤ i, j < |mat|) →L 0 ≤ mat[i][j] }
```

1.4. Ejercicio 4

```
proc cantidadCaminosNSaltos (inout conexion: seq<seq<Z>>, in n: Z)
  requiere {1 ≤ n}
  requiere {esCuadrada(conexion) ∧L filaIgualColumna(conexion)}
  requiere {(∀i, j : Z) (0 ≤ i, j < |conexion|) →L conexion[i][j] ∈ [0, 1]}
  requiere {conexion = C0}
  asegura {(∃sec : seq<seq<seq<Z>>>) (|sec| = n) ∧ sec[0] = C0 ∧ conexion = sec[|sec|-1] ↔
  (∀i : Z) (1 ≤ i < |sec|) →L esCuadrada(sec[i]) ∧ |sec[i]| = |conexion| →L
  esLaMultiplicacion(sec[i], C0, sec[i-1]))}

pred esLaMultiplicacion (in mat: seq<seq<seq<Z>>>, in mat0 : seq<seq<seq<Z>>>, in mat1 : seq<seq<seq<Z>>>) { (∀i, j : Z) (0 ≤ i, j < |mat|) →L mat[i][j] = ∑k=0|mat|-1 mat0[i][k] * mat1[k][j] }
```

1.5. Ejercicio 5

```

proc caminoMinimo (in origen: Z, in destino: Z, in distancias: seq⟨seq⟨Z⟩⟩) : seq⟨Z⟩
  requiere {esCuadrada(distancias) ∧L filaIgualColumna(distancias)}
  requiere {0 ≤ destino, origen < |distancias|}
  requiere {matrizTodosPositivos(distancias)}
  asegura {(∃s1 : seq⟨Z⟩) (|s1| > 1) ∧ res = s1 ↔ (∀s2 : seq⟨Z⟩) (|s2| > 1) →L
    (secuenciaEsCamino(distancias, s1, origen, destino) ∧ secuenciaEsCamino(distancias, s2, origen, destino) ∧
    (longitudCamino(s1, distancias) ≤ longitudCamino(s2, distancias)))}
  asegura {res = [] ↔ (∀s : seq⟨Z⟩) (¬secuenciaEsCamino(distancias, s, origen, destino))}

aux longitudCamino (in sec: seq⟨Z⟩, in distancias: seq⟨seq⟨Z⟩⟩) : Z = ∑i=1|sec|-1 distancias[sec[i]][sec[i-1]] ;

```

2. Punto 2

2.1. Ejercicio 1

Demostramos que la implementación es correcta con respecto a la especificación dada mediante teorema de invariante y teorema de terminación.

Por teorema del invariante primero debemos demostrar los siguientes puntos:

- $P_c \longrightarrow I$
- $\{I \wedge B\} S \{I\}$
- $(I \wedge \neg B) \longrightarrow Q_c$

Elegimos nuestra invariante.

$I \equiv 0 \leq i \leq |\text{ciudades}| \wedge \sum_{j=0}^{i-1} \text{ciudades}[j].\text{habitantes} = \text{res}$

Primer paso Probamos primero la implicación de la precondition del ciclo hacia el invariante:

$P_c \longrightarrow I$

$$\text{res} = 0 \wedge i = 0 \longrightarrow 0 \leq i \leq |\text{ciudades}| \wedge \sum_{j=0}^{i-1} \text{ciudades}[j].\text{habitantes} = \text{res}$$

$$0 \leq 0 \leq |\text{ciudades}| \wedge \sum_{j=0}^{0-1} \text{ciudades}[j].\text{habitantes} = 0$$

$$\text{True} \wedge \text{True}$$

$$\text{True}$$

Segundo paso Ahora probamos que vale la siguiente tripla de Hoare:

$$\{I \wedge B\} S \{I\}$$

Para probar que esto sea verdadero se debe cumplir $\{I \wedge B\} \longrightarrow wp(S, I)$

Hacemos uso del axioma para calcular $wp(S, I)$.

$$wp(S, I) \equiv wp(S_1, wp(S_2, I))$$

$$wp(S_2, I) \equiv def(S_2) \wedge_L I_{i:=i+1}^i$$

$$wp(S_2, I) \equiv True \wedge 0 \leq i + 1 \leq |ciudades| \wedge_L \sum_{j=0}^i ciudades[j].habitantes = res$$

Terminamos de definir el wp de S2:

$$wp(S_2, I) \equiv 0 \leq i + 1 \leq |ciudades| \wedge_L \sum_{j=0}^i ciudades[j].habitantes = res$$

Ahora definimos el wp de S1:

$$wp(S_1, wp(S_2, I)) \equiv wp(res := res + ciudades[i].habitantes, 0 \leq i + 1 \leq |ciudades| \wedge_L \sum_{j=0}^i ciudades[j].habitantes = res)$$

$$wp(S_1, wp(S_2, I)) \equiv def(res := res + ciudades[i].habitantes) \wedge_L I_{res:=res+ciudades[i].habitantes}^{res}$$

$$wp(S_1, wp(S_2, I)) \equiv 0 \leq i < |ciudades| \wedge_L 0 \leq i + 1 \leq |ciudades| \wedge_L \sum_{j=0}^i ciudades[j].habitantes = res + ciudades[i].habitantes$$

$$wp(S_1, wp(S_2, I)) \equiv 0 \leq i < |ciudades| \wedge_L \sum_{j=0}^i ciudades[j].habitantes = res + ciudades[i].habitantes$$

$$wp(S_1, wp(S_2, I)) \equiv 0 \leq i < |ciudades| \wedge_L \sum_{j=0}^i ciudades[j].habitantes - ciudades[i].habitantes = res$$

Ahora queda definido el wp de S:

$$wp(S, I) \equiv 0 \leq i < |ciudades| \wedge_L \sum_{j=0}^{i-1} ciudades[j].habitantes = res$$

Ahora veo la implicacion del invariante y la guarda hacia $wp(S, I)$:

$$\{I \wedge B\} \longrightarrow wp(S, I) \equiv$$

$$0 \leq i < |ciudades| \wedge 0 \leq i \leq |ciudades| \wedge_L \sum_{j=0}^{i-1} ciudades[j].habitantes = res \longrightarrow$$

$$0 \leq i < |ciudades| \wedge_L \sum_{j=0}^{i-1} ciudades[j].habitantes = res$$

Se cancelan los terminos y queda:

$$0 \leq i \leq |ciudades| \longrightarrow True$$

$$True$$

Tercer paso Ahora probamos que vale la siguiente implicación: $(I \wedge \neg B) \longrightarrow Q_c$

$$0 \leq i \leq |\text{ciudades}| \wedge_L \sum_{j=0}^{i-1} \text{ciudades}[j].\text{habitantes} = \text{res} \wedge i \geq |\text{ciudades}| \longrightarrow$$

$$\sum_{j=0}^{|\text{ciudades}|-1} \text{ciudades}[j].\text{habitantes} = \text{res} \wedge i = |\text{ciudades}|$$

i esta entre las longitudes de ciudades y se cancelan las sumatorias

True

Ahora por teorema de terminación debemos demostrar que la ejecucion del ciclo siempre termina, nuestra función variante:

$F_v = |\text{ciudades}| - i - 1$

Para probar que esto sea verdadero se debe cumplir lo siguiente.

- $\{I \wedge B \wedge F_v = v_0\} S \{F_v < v_0\}$
- $(I \wedge F_v \leq 0) \longrightarrow \neg B$

Primer paso Probamos la primera implicación:

$\{I \wedge B \wedge F_v = v_0\} S \{F_v < v_0\}$

$$\{0 \leq i \leq |\text{ciudades}| \wedge_L \sum_{j=0}^{i-1} \text{ciudades}[j].\text{habitantes} = \text{res} \wedge 0 \leq i \leq |\text{ciudades}| \wedge F_v = |\text{ciudades}| - i - 1\} \longrightarrow$$

$$wp(S, F_v < v_0)$$

Calculamos la wp para probar $\{I \wedge B \wedge F_v = v_0\} \longrightarrow wp(S, \{F_v < v_0\})$

$$wp(S1, wp(S2, F_v < v_0)) \equiv$$

$$wp(S1, wp(i := i + 1, |\text{ciudades}| - i - 1 < v_0)) \equiv$$

$$def(i := i + 1) \wedge |\text{ciudades}| - (i + 1) - 1 < v_0 \equiv$$

$$wp(S1, True \wedge |\text{ciudades}| - i - 2 < v_0) \equiv$$

$$wp(\text{res} := \text{res} + \text{ciudades}[i].\text{habitantes}, |\text{ciudades}| - i - 2 < v_0) =$$

$$def(\text{res} := \text{res} + \text{ciudades}[i].\text{habitantes}) \wedge_L |\text{ciudades}| - i - 2 < v_0 \equiv$$

$$wp(S, F_v < v_0) \equiv 0 \leq i < |\text{ciudades}| \wedge_L |\text{ciudades}| - i - 2 < v_0$$

Finalmente la implicación nos queda de la forma

$\{I \wedge B \wedge F_v = v_0\} \longrightarrow wp(S, F_v < v_0)$

$$\{0 \leq i \leq |\text{ciudades}| \wedge_L \sum_{j=0}^{i-1} \text{ciudades}[j].\text{habitantes} = \text{res} \wedge 0 \leq i < |\text{ciudades}| \wedge |\text{ciudades}| - i - 1 = v_0 \longrightarrow$$

$$0 \leq i < |\text{ciudades}| \wedge_L |\text{ciudades}| - i - 2 < v_0 \equiv$$

$$|\text{ciudades}| - i - 2 < |\text{ciudades}| - i - 1 \equiv True$$

Segundo paso Probamos la segunda implicación:

$(I \wedge F_v \leq 0) \longrightarrow \neg B$

$$0 \leq i \leq |\text{ciudades}| \wedge_L \text{res} = \sum_{j=0}^{i-1} \text{ciudades}[j].\text{habitantes} \wedge_L |\text{ciudades}| - i - 1 \leq 0 \longrightarrow i \geq |\text{ciudades}| \equiv$$

$$0 \leq i \leq |\mathbf{ciudades}| \wedge_L |\mathbf{ciudades}| \leq i + 1 \longrightarrow i \geq |\mathbf{ciudades}| \equiv$$

$$i = |\mathbf{ciudades}| \longrightarrow i \geq |\mathbf{ciudades}| \equiv$$

$$True$$

2.2. Ejercicio 2

Teniendo en cuenta la correctitud del programa demostrada en el 2.1, sabemos que el valor devuelto por el programa, siempre que se cumpla la precondition, tendrá la pinta de:

$$\mathbf{res} = \sum_{j=0}^{|\mathbf{ciudades}|-1} \mathbf{ciudades}[j].\mathbf{habitantes}$$

En el procedimiento, se especifica que el parámetro de entrada es de tipo IN, por lo tanto en la precondition y la postcondición, el parámetro *ciudades* mantendrá las mismas características, entre ellas que sus elementos son todos mayores o iguales a 0. Por lo tanto, al tomar un elemento de *ciudades*, obligatoriamente será menor o igual a la sumatoria del total de elementos de *ciudades*, es decir:

$$(\forall i : Z) (0 \leq i < |\mathbf{ciudades}|) \longrightarrow_L$$

$$\mathbf{ciudades}[i].\mathbf{habitantes} \leq \sum_{j=0}^{|\mathbf{ciudades}|-1} \mathbf{ciudades}[j].\mathbf{habitantes}$$

Ahora bien, según la postcondición sabemos que *res* es igual la sumatoria de todos sus elementos, entre los cuales sabemos que al menos alguno de ellos es mayor a 50.000, por lo tanto:

$$(\exists k : Z) (0 \leq k < |\mathbf{ciudades}|) \wedge_L$$

$$50,000 < \mathbf{ciudades}[k].\mathbf{habitantes}$$

Podemos afirmar entonces que:

$$50,000 < \mathbf{ciudades}[k].\mathbf{habitantes} \leq \sum_{j=0}^{i-1} \mathbf{ciudades}[j].\mathbf{habitantes}$$

Y por transitividad concluimos que:

$$50,000 < \sum_{j=0}^{i-1} \mathbf{ciudades}[j].\mathbf{habitantes}$$