



Trabajo práctico 1

Especificación y WP

12 de septiembre de 2024

Algoritmos y Estructura de Datos

SinGrupo4

Integrante	LU	Correo electrónico
Algañaraz, Franco	001/01	francoarga10@gmail.com
Illescas, Marcos	390/14	marcosillescas90@gmail.com
Bahamonde, Matias	694/21	matubaham@gmail.com
Marión, Ian Pablo	004/01	ianfrodin@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. Especificación

```
pred habitantesPositivos (in s: seq⟨Ciudad⟩) {  
  (∀j : ℤ)(0 ≤ j < |s| →L s[j].habitantes ≥ 0)  
}  
pred noCiudadesRepetidas (in s: seq⟨Ciudad⟩) {  
  (∀i : ℤ)(∀j : ℤ)(0 ≤ i < j < |s| →L s[i].nombre ≠ s[j].nombre)  
}
```

1.1. grandesCiudades

```
proc grandesCiudades (in ciudades: seq⟨Ciudad⟩) : seq⟨Ciudad⟩  
  requiere {habitantesPositivos(ciudades) ∧ noCiudadesRepetidas(ciudades)}  
  asegura {|res| = CantidadCiudadesMayor50000(ciudades) ∧  
    (∀i : ℤ)(0 ≤ i < |res| →L (∃j : ℤ)(0 ≤ j < |ciudades| ∧L (res[i] = ciudades[j] ∧ ciudades[j].habitantes > 50000))))}  
aux CantidadCiudadesMayor50000 (in s: seq⟨Ciudad⟩) : ℤ =  
  ∑i=0|s|-1 if s[i].habitantes > 50000 then 1 else 0 fi;
```

1.2. sumaDeHabitantes

```
proc sumaDeHabitantes (in menoresDeCiudades: seq⟨Ciudad⟩, in mayoresDeCiudades: seq⟨Ciudad⟩) : seq⟨Ciudad⟩  
  requiere {|menoresDeCiudades| = |mayoresDeCiudades| ∧  
    mismosNombres(menoresDeCiudades, mayoresDeCiudades) ∧  
    habitantesPositivos(menoresDeCiudades) ∧ habitantesPositivos(mayoresDeCiudades) ∧  
    noCiudadesRepetidas(menoresDeCiudades) ∧ noCiudadesRepetidas(mayoresDeCiudades)}  
  asegura {|res| = |menoresDeCiudades| ∧  
    (∀i : ℤ)(0 ≤ i < |res| →L (∃j : ℤ)(∃k : ℤ)(0 ≤ j < k < |menoresDeCiudad| ∧L  
    siCoincidenNombresSumoHabitantes(res, menoresDeCiudades, mayoresDeCiudades, i, j, k))))}  
pred mismosNombres (in s: seq⟨Ciudad⟩, in t: seq⟨Ciudad⟩) {  
  (∀i : ℤ)(0 ≤ i < |s| →L (∃j : ℤ)(0 ≤ j < |t| ∧L s[i].nombre = t[j].nombre))  
}  
pred siCoincidenNombresSumoHabitantes (in s: seq⟨Ciudad⟩, in t: seq⟨Ciudad⟩, in r: seq⟨Ciudad⟩, in i: ℤ, in j: ℤ, in k: ℤ) {  
  (s[i].nombre = t[j].nombre ∧ s[i].nombre = r[k].nombre) → s[i].habitantes = t[j].habitantes + r[k].habitantes  
}
```

1.3. hayCamino

```
proc hayCamino (in distancias : seq⟨seq⟨ℤ⟩⟩, in desde : ℤ, in hasta : ℤ) : Bool  
  requiere {0 ≤ desde < |distancias| ∧ 0 ≤ hasta < |distancias|}  
  asegura {res = true ⇔ existeCamino(distancia, desde, hasta)}  
pred existeCamino (in distancias : seq⟨seq⟨ℤ⟩⟩, in desde : ℤ, in hasta : ℤ) {  
  (∃camino : seq⟨ℤ⟩) (|camino| ≥ 2 ∧ camino[0] = desde ∧ camino[|camino| - 1] = hasta ∧ (∀i : ℤ)(0 ≤ i < |camino| - 1 →  
    distancias[camino[i]][camino[i + 1]] > 0))  
}
```

1.4. cantidadCaminosNSaltos

```
proc cantidadCaminosNSaltos (in conexion : seq⟨seq⟨ℤ⟩⟩, in n : ℤ) : seq⟨ℤ⟩  
  requiere {n ≥ 1 ∧ conexion = C0}  
  asegura {conexion = C0n}
```

1.5. caminoMínimo

```
proc caminoMinimo (in origen : ℤ, in destino : ℤ, in distancias : seq⟨seq⟨ℤ⟩⟩) : seq⟨ℤ⟩  
  requiere {0 ≤ origen < |distancias| ∧ 0 ≤ destino < |distancias|}  
  asegura {(res = [ ] ⇔ existeCamino(distancias, origen, destino) ∨ origen = destino) ∧ (res ≠ [ ] ⇒  
    (∀camino : seq⟨ℤ⟩)(camino[0] = origen ∧ camino[|camino| - 1] = destino ∧ sumaDistancias(res, distancias) ≤  
    sumaDistancias(camino, distancias)))}
```

aux sumaDistancia (in camino : seq⟨ℤ⟩, in distancias : seq⟨seq⟨ℤ⟩⟩) : ℤ = $\sum_{i=0}^{|\text{camino}|-2} \text{distancias}[\text{camino}[i]][\text{camino}[i+1]]$;

2. Demostraciones de correctitud

2.1. Demostrar que la implementación es correcta con respecto a la especificación.

```
proc poblacionTotal (in ciudades: seq⟨Ciudad⟩) : ℤ
  requiere { (∃j : ℤ)(0 ≤ j < |ciudades| ∧L ciudades[j].habitantes > 50000) ∧
    (∀k : ℤ)(0 ≤ k < |ciudades| →L ciudades[k].habitantes ≥ 0) ∧
    (∀m : ℤ)(∀n : ℤ)(0 ≤ m < n < |ciudades| →L ciudades[m].nombre ≠ ciudades[n].nombre) }
  asegura { res =  $\sum_{i=0}^{|\text{ciudades}|-1} \text{ciudades}[i].\text{habitantes}$  }
```

```
S1 : res = 0
S2 : i = 0
while (i < ciudades.length) do
S3 : res = res + ciudades[i].habitantes
S4 : i = i + 1
endwhile
```

```
P ≡ A ∧ B ∧ C
A ≡ (∃l : ℤ)(0 ≤ l < |ciudades| ∧L ciudades[l].habitantes > 50000)
B ≡ (∀k : ℤ)(0 ≤ k < |ciudades| →L ciudades[k].habitantes ≥ 0)
C ≡ (∀m : ℤ)(∀n : ℤ)(0 ≤ m < n < |ciudades| →L ciudades[m].nombre ≠ ciudades[n].nombre)
```

$Pc \equiv res = 0 \wedge i = 0 \wedge A \wedge B \wedge C$

$B \equiv i < |\text{ciudades}|$

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

$f_v \equiv |\text{ciudades}| - i$

Como el programa finaliza al terminar el ciclo, podemos asumir que:

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

Para demostrar la correctitud del programa basta ver que son válidas las siguientes triplas de Hoare:

$\{Pc\} S_3; S_4 \{Qc \equiv Q\}$ y $\{P\} S_1; S_2 \{Pc\}$

2.1.1. $Pc \implies wp(S_3; S_4, Qc)$ (Teorema del Invariante y Terminación)

2.1.1.1 $Pc \implies I$

$Pc \equiv res = 0 \wedge i = 0 \wedge A \wedge B \wedge C \implies 0 \leq 0 \leq |\text{ciudades}| \wedge_L 0 = \sum_{j=0}^{-1} \text{ciudades}[j].\text{habitantes} \equiv$

$0 \leq |\text{ciudades}| \wedge_L 0 = 0 \equiv$

$True \wedge_L True \equiv$

$True$

$Pc \implies I$

2.1.1.2 $I \wedge B \implies wp(S_3; S_4, I)$

$wp(S_3; S_4, I) \equiv wp(res = res + \text{ciudades}[i].\text{habitantes}, wp(i = i + 1, I)) \equiv$

$wp(res = res + \text{ciudades}[i].\text{habitantes}, \text{def}(i + 1) \wedge_L I_{i+1}^i) \equiv$

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

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

$$def(res + ciudades.[i].habitantes) \wedge_L (I')_{res+ciudades.[i].habitantes}^{res} \equiv$$

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

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

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

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

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

$$True \wedge_L True \equiv$$

$$True$$

$$I \wedge B \implies wp(S_3; S_4, I)$$

2.1.1.3 $I \wedge \neg B \implies Qc$

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

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

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

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

$$True \wedge_L True \equiv$$

$$True$$

$$I \wedge \neg B \implies Qc$$

2.1.1.4 $I \wedge B \wedge v_0 = f_v \implies wp(S_3; S_4, f_v < v_0)$

$$wp(S_3; S_4, f_v < v_0) \equiv wp(res = res + ciudades[i].habitantes, wp(i = i + 1, |ciudades| - i < v_0)) \equiv$$

$$wp(res = res + ciudades[i].habitantes, def(i + 1) \wedge_L (|ciudades| - i < v_0)_{i+1}^i) \equiv$$

$$wp(res = res + ciudades[i].habitantes, True \wedge_L |ciudades| - i - 1 < v_0) \equiv$$

$$def(res + ciudades[i].habitante) \wedge_L (|ciudades| - i - 1 < v_0)_{res+ciudades[i].habitante}^{res} \equiv$$

$$0 \leq i < |ciudades| \wedge_L |ciudades| - i - 1 < v_0 \equiv$$

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

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

$$0 \leq i < |ciudades| \wedge_L |ciudades| - i - 1 < |ciudades| - i \equiv$$

$$True \wedge_L True \equiv$$

$$True$$

$$I \wedge B \wedge v_0 = f_v \implies wp(S_3; S_4, f_v < v_0)$$

2.1.1.5 $I \wedge f_v \leq 0 \implies \neg B$

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

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

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

$$i \geq |ciudades| \equiv$$

$$\begin{aligned} & True \\ I \wedge f_v \leq 0 & \implies \neg B \end{aligned}$$

2.1.2. $P \implies wp(S_1; S_2, Pc)$

$$\begin{aligned} wp(S_1; S_2, Pc) &\equiv wp(res = 0, wp(i = 0, Pc)) \equiv wp(res = 0, def(0) \wedge_L Pc_0^i) \equiv \\ wp(res = 0, True \wedge_L res = 0 \wedge 0 = 0 \wedge A \wedge B \wedge C) &\equiv \\ def(0) \wedge_L (res = 0 \wedge True \wedge A \wedge B \wedge C)^{res}_0 &\equiv \\ True \wedge_L 0 = 0 \wedge A \wedge B \wedge C \equiv True \wedge A \wedge B \wedge C &\equiv A \wedge B \wedge C \end{aligned}$$

$$P \equiv A \wedge B \wedge C \implies A \wedge B \wedge C \equiv True \wedge True \wedge True \equiv True$$

$$P \implies wp(S_1; S_2, Pc)$$

El programa es correcto con respecto a su especificación.

2.2. Demostrar que el valor devuelto es mayor a 50.000.

Para demostrar que el valor devuelto es mayor a 50.000, cambiamos la Postcondición Q del programa acordemente y vemos si el programa sigue siendo correcto con respecto a su especificación.

Consideramos P, B, Pc, f_v igual que en el punto anterior, y

$$Q \equiv Qc \equiv res = \sum_{j=0}^{|ciudades|-1} ciudades[j].habitantes \wedge res > 50000$$

$$A \equiv (\exists l : \mathbb{Z})(0 \leq l < |ciudades| \wedge_L ciudades[l].habitantes > 50000)$$

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