



# Trabajo Práctico 1: Especificación y WP

”En Búsqueda del Camino”

13 de septiembre de 2024

Algoritmos y Estructuras de Datos

## Grupo puntoJava

Integrante	LU	Correo electrónico
Gremes, Juan Ignacio	21/24	juanigremes@gmail.com
Anllo, Francisco	209/24	anllo.francisco@gmail.com
Naddeo, Matias	651/24	email3@dominio.com
Gutierrez, Marco	167/24	marcoantonio-gutierrez02@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. Especificación

## 1.1. grandesCiudades:

A partir de una lista de ciudades, devuelve aquellas que tienen más de 50.000 habitantes.

```
proc grandesCiudades (in ciudades : seq⟨String × ℤ⟩) : seq⟨String × ℤ⟩
  requiere {True}
  asegura {(∀i : ℤ) ((0 ≤ i < |res|) →L ((res[i] ∈ ciudades) ∧L (res[i]1 > 50,000)))}
  asegura {(∀j : ℤ) (((0 ≤ j < |ciudades|) ∧L (ciudades[j]1 > 50,000)) →L (ciudades[j] ∈ res))}
```

## 1.2. sumaDeHabitantes:

Por cuestiones de planificación urbana, las ciudades registran sus habitantes mayores de edad por un lado y menores de edad por el otro.

Dadas dos listas de ciudades del mismo largo con los mismos nombres, una con sus habitantes mayores y otra con sus habitantes menores, este procedimiento debe devolver una lista de ciudades con la cantidad total de sus habitantes.

```
proc sumaDeHabitantes (in menoresDeCiudades : seq⟨String × ℤ⟩, in mayoresDeCiudades : seq⟨String × ℤ⟩) : seq⟨String × ℤ⟩
  requiere {|menoresDeCiudades| = |mayoresDeCiudades|}
  requiere {(∀i : ℤ) ((0 ≤ i < |menoresDeCiudades|) →L (enLaOtraEdad(menoresDeCiudades[i]0,
    mayoresDeCiudades))))}
  asegura {|res| = |menoresDeCiudades|}
  asegura {(∀x, y, z : ℤ) (((0 ≤ x < |res|) ∧ (0 ≤ y < |res|) ∧ (0 ≤ z < |res|) ∧L (menoresDeCiudades[x]0 =
    mayoresDeCiudades[y]0 = res[z]0)) →L (res[z]1 = menoresDeCiudades[x]1 + mayoresDeCiudades[y]1))}
  asegura {(∀i : ℤ) ((0 ≤ i < |res|) →L (enLaOtraEdad(res[i]0, menoresDeCiudades))))}

pred enLaOtraEdad (nombreCiudad : String, listaCiudades : seq⟨String × ℤ⟩) {
  (∃j : ℤ) ((0 ≤ j < |listaCiudades|) ∧L (nombreCiudad = listaCiudades[j]0))
}
```

## 1.3. hayCamino:

Un mapa de ciudades está conformada por ciudades y caminos que unen a algunas de ellas. A partir de este mapa, podemos definir las distancias entre ciudades como una matriz donde cada celda i, j representa la distancia entre la ciudad i y la ciudad j. Una distancia de 0 equivale a no haber camino entre i y j. Notar que la distancia de una ciudad hacia sí misma es cero y la distancia entre A y B es la misma que entre B y A.

Dadas dos ciudades y una matriz de distancias, se pide determinar si existe un camino entre ambas ciudades.

```
proc hayCamino (in distancias : seq⟨seq⟨ℤ⟩⟩, in desde : ℤ, in hasta : ℤ) : Bool
  requiere {(0 ≤ desde, hasta < |distancias|)}
  requiere {(∀i, j : ℤ) ((0 ≤ i, j < |distancias|) →L (|distancias| = |distancias[i]| ∧L 0 ≤ distancias[i][j]))}
  asegura {res = True ⇔ (∃s : seq⟨ℤ⟩) (esCamino(distancias, desde, hasta, s))}

pred esCamino (in distancias : seq⟨seq⟨ℤ⟩⟩, in desde : ℤ, in hasta : ℤ, in camino : seq⟨ℤ⟩) {
  (|camino| > 1) ∧L (camino[0] = desde ∧ camino[|camino| - 1] = hasta) ∧ (∀i : ℤ) ((0 ≤ i < |camino| - 1) →L ((0 ≤
    camino[i] < |distancias|) ∧L (hayCaminoDirecto(camino[i], camino[i + 1], distancias))))
}

pred hayCaminoDirecto (c1 : ℤ, c2 : ℤ, distancias : seq⟨seq⟨ℤ⟩⟩) {
  distancias[c1][c2] ≠ 0
}
```

## 1.4. cantidadCaminosNSaltos:

Dentro del contexto de redes informáticas, nos interesa contar la cantidad de “saltos” que realizan los paquetes de datos, donde un salto se define como pasar por un nodo. Así como definimos la matriz de distancias, podemos definir la matriz de conexión entre nodos, donde cada celda  $i, j$  tiene un 1 si hay un único camino a un salto de distancia entre el nodo  $i$  y el nodo  $j$ , y un 0 en caso contrario. En este caso, se trata de una matriz de conexión de orden 1, ya que indica cuáles pares de nodos poseen 1 camino entre ellos a 1 salto de distancia.

Dada la matriz de conexión de orden 1, este procedimiento debe obtener aquella de orden  $n$  que indica cuántos caminos de  $n$  saltos hay entre los distintos nodos. Notar que la multiplicación de una matriz de conexión de orden 1 consigo misma nos da la matriz de conexión de orden 2, y así sucesivamente.

```

proc cantidadCaminosNSaltos (inout conexión : seq<seq<Z>>, in n : Z) : seq<seq<Z>>
  requiere {conexión = CONEXION0}
  requiere {(∀i : Z) ((0 ≤ i < |conexión|) →L (|conexión| = |conexión[i]|))}
  requiere {(∀i, j : Z) ((0 ≤ i, j < |conexión|) →L (0 ≤ conexión[i][j]))}
  asegura {(∃s : seq<seq<seq<Z>>>) ((s[0] = CONEXION0) ∧L (matricesConsecutivas(s)) ∧L (conexión = s[|s| - 1]))}

pred matricesConsecutivas (secuenciaDeMatrices : seq<seq<seq<Z>>>) {
  (∀i : Z) ((0 ≤ i < |secuenciaDeMatrices| - 1) →L (multipDeMatrices(secuenciaDeMatrices[i + 1], secuenciaDeMatrices[i], s)))
}

pred multipDeMatrices (in matriz1 : seq<seq<Z>>, in matriz2 : seq<seq<Z>>, in matrizOriginal : seq<seq<Z>>) {
  (∀x, y : Z) ((0 ≤ x, y < |matriz1|) →L (matriz1[x][y] = filaXcolumna(x, y, matriz2, matrizOriginal)))
}

aux fila X columna (in x : Z, in y : Z, in matriz1 : seq<seq<Z>>, in matriz2 : seq<seq<Z>>) : Z =  $\sum_{k=0}^{|matriz1|-1} (matriz1[x][k] * matriz2[k][y])$ ;

```

## 1.5. caminoMínimo

Dada una matriz de distancias, una ciudad de origen y una ciudad de destino, este procedimiento debe devolver la lista de ciudades que conforman el camino más corto entre ambas. En caso de no existir un camino, se debe devolver una lista vacía.

```

proc caminoMinimo (in distancias : seq<seq<Z>>, in desde : Z, in hasta : Z) : seq<Z>
  requiere {0 ≤ desde, hasta < |distancias|}
  requiere {(∀i, j : Z) (0 ≤ i, j < |distancias| →L (|distancias| = |distancias[i]| ∧L 0 ≤ distancias[i][j]))}
  asegura {res = {} ⇔ (∀s : seq<Z>) (¬esCamino(distancias, desde, hasta, s))}
  asegura {(∀s : seq<Z>) ((esCamino(distancias, desde, hasta, s)) →L (distancia(distancias, s) ≥ distancia(distancias, res)))}

aux distancia (in distancias : seq<seq<Z>>, in camino : seq<Z>) : Z =  $\sum_{k=0}^{|s|-2} distancias[camino[k]][camino[k + 1]]$ ;

```

## 2. Demostraciones de Correctitud

La función poblaciónTotal recibe una lista de ciudades donde al menos una de ellas es grande (es decir, supera los 50.000 habitantes) y devuelve la cantidad total de habitantes. Dada la siguiente especificación:

```

proc poblaciónTotal (in ciudades : seq<String × Z>) : Z
  requiere {(∃i : Z) ((0 ≤ i < |ciudades|) ∧L (ciudades[i].habitantes > 50,000)) ∧
  (∀i : Z) ((0 ≤ i < |ciudades|) →L (ciudades[i].habitantes ≥ 0)) ∧
  (∀i, j : Z) ((0 ≤ i < j < |ciudades|) →L (ciudades[i].nombre ≠ ciudades[j].nombre))}
  asegura {res =  $\sum_{i=0}^{|ciudades|-1} ciudades[i].habitantes$ }

```

Con la siguiente implementación:

```

res = 0
i = 0
while ( i < ciudades.length) do
  res = res + ciudades[i].habitantes
  i = i + 1
endwhile

```

### 1. Demostrar que la implementación es correcta con respecto a la especificación.

Para poder asegurar que la implementación es correcta, debemos probar que se cumple la tripla de Hoare  $[P] S [Q]$ . Es decir,  $P \rightarrow_L \text{wp}(S, Q)$ . En primer lugar, intentaremos identificar esta precondition  $P$  y post condición  $Q$ . La post condición  $Q$  es el estado que buscaremos que el programa se encuentre una vez que finalice. De acuerdo a la especificación, es posible concluir que:  $Q = \sum_{i=0}^{|ciudades|-1} ciudades[i].habitantes$

Como el programa termina en el ciclo "while", es válido afirmar que la post condición del programa será la misma que la del ciclo. En otras palabras,  $Q = Q_c$ . Es imperativo, entonces, comprobar la correctitud del ciclo, y para ello utilizaremos el Teorema del Invariante y el Teorema de Terminación.

Veamos, en primer lugar, cuál es la precondition del ciclo. Sabemos que esta será la precondition del programa, más cualquier instrucción hasta llegar al comienzo del ciclo. Entonces, podemos decir que, como  $P$  es todas las condiciones establecidas en los *requiere*, entonces  $P_c$  lo será también, sumadas las dos instrucciones antes de llegar al ciclo. Por lo tanto,  $P_c = P \wedge i = 0 \wedge res = 0$

Proponemos, entonces, el siguiente invariante:  $I = 0 \leq i \leq |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].habitantes$ . Veamos ahora si cumple las tres condiciones necesarias:

1.  $P_c \rightarrow I$ :

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

Como la sumatoria se encuentra fuera de rango, el total es igual a 0. Se cumple, entonces, que  $res = 0$ . Verificamos que esta condición es verdadera; veamos las otras dos.

2.  $I \wedge \neg B \rightarrow Q_c$ :

Llamamos  $B$  a la guarda del "while". En este caso, la guarda es  $i < ciudades.length$ , por lo que la negación de  $B$  sería  $i \geq |ciudades|$ . Ahora que tenemos bien definido todo, veamos si esto se cumple:

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

Si condensamos estas condiciones, es claro que...

$$I \wedge \neg B \rightarrow (i = |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].habitantes) \rightarrow res = \sum_{j=0}^{|ciudades|-1} ciudades[j].habitantes$$

¡Que es exactamente lo que establece  $Q_c$ ! Veamos la última condición.

3.  $[I \wedge B] S [I]$

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

$$\text{Condensado, esto es lo mismo que decir } (0 \leq i < |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].habitantes) \rightarrow_L \text{wp}(S, I)$$

Para esto, llamaremos  $S1$  a la primera instrucción dentro del ciclo, y  $S2$  a la segunda. Por lo tanto,  $S1 = (res = res + ciudades[i].habitantes)$  y  $S2 = (i = i + 1)$

$$\text{wp}(S, I) = \text{wp}(S1, \text{wp}(S2, I)) = \text{wp}(res = res + ciudades[i].habitantes, \text{wp}(S2, I))$$

$$\text{wp}(S2, I) = (i := i + 1, 0 \leq i \leq |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].habitantes)$$

Utilizando el Axioma 1 (Asignación), reemplazaremos el valor de  $i$  en cada una de sus apariciones libres, que en este caso, son todas:

$$\text{wp}(S2, I) = \text{def } (i + 1) \wedge_L (0 \leq i + 1 \leq |ciudades| \wedge_L res = \sum_{j=0}^{i+1-1} ciudades[j].habitantes)$$

Como  $i+1$  no trae ninguna restricción, su definición es igual a *True*. Nos queda, entonces...

$$\text{wp}(S2, I) = 0 \leq i < |ciudades| \wedge_L res = \sum_{j=0}^i ciudades[j].habitantes$$

Retomando con lo anterior, obtenemos que:

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

Nuevamente, utilizaremos el Axioma 1. La expresion queda de la siguiente manera:

$$\text{def } (\text{res}) \wedge_L \text{def } (\text{ciudades}[i]) \wedge_L \text{res} + \text{ciudades}[i].\text{habitantes} = \sum_{j=0}^i \text{ciudades}[j].\text{habitantes}$$

Como  $\text{res}$  no tiene restricciones, esa definición es *True*. Sin embargo,  $\text{ciudades}[i]$  requiere que  $i$  esté en rango de la lista.

Por otro lado, podemos decir que  $\sum_{j=0}^i \text{ciudades}[j].\text{habitantes} = \sum_{j=0}^{i-1} (\text{ciudades}[j].\text{habitantes}) + \text{ciudades}[i].\text{habitantes}$

Veamos cómo sigue:

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

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

Recapitulando...

$(0 \leq i < |\text{ciudades}| \wedge_L \text{res} = \sum_{j=0}^{i-1} \text{ciudades}[j].\text{habitantes}) \longrightarrow_L \text{wp}(\text{S}, \text{I})$  es verdadero, porque son exactamente el mismo predicado. Por lo tanto, el ciclo es parcialmente correcto. Falta comprobar si siempre termina, que probaremos a través del Teorema de Terminación.

Proponemos la Función Variante ( $\text{fv}$ ) =  $|\text{ciudades}| - i$ . Veamos si cumple las condiciones:

$$1. \text{I} \wedge \text{fv} \leq 0 \longrightarrow \neg B$$

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

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

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

Que es verdadero, ya que  $i = |\text{ciudades}| \longrightarrow i \geq \text{ciudades}$ . Veamos la siguiente condición.

$$2. .[\text{I} \wedge B \wedge \text{V0} = \text{fv}] \text{S} [\text{fv} < \text{V0}]$$

$$\text{I} \wedge_L B \wedge_L \text{V0} = |\text{ciudades}| - i \longrightarrow \text{wp}(\text{S}, \text{fv} < \text{V0})$$

Usando  $\text{S}$ , dividido en  $\text{S1}$  y  $\text{S2}$  como fue explicado anteriormente, podemos decir que:

$$\text{wp}(\text{S}, \text{fv} < \text{V0}) = \text{wp}(\text{res} = \text{res} + \text{ciudades}[i].\text{habitantes}, \text{wp}(\text{S2}, \text{fv} < \text{V0}))$$

$\text{wp}(\text{S2}, \text{fv} < \text{V0}) = \text{wp}(i := i + 1, |\text{ciudades}| - i < \text{V0})$  Usando el Axioma 1 obtenemos:  $|\text{ciudades}| - i - 1 < \text{V0}$ , ya que  $(i + 1)$  ya está definido.

$\text{wp}(\text{S}, \text{fv} < \text{V0}) = \text{wp}(\text{res} = \text{res} + \text{ciudades}[i].\text{habitantes}, |\text{ciudades}| - i - 1 < \text{V0})$ . Usando el axioma 1, obtenemos:  $\text{def}(s[i]) \wedge_L |\text{ciudades}| - i - 1 < \text{V0}$ , porque  $\text{res}$  ya está definido.

$(0 \leq i \leq |\text{ciudades}| \wedge_L |\text{ciudades}| - i - 1 < |\text{ciudades}| - i) \longrightarrow \text{fv} < \text{V0}$ . Se cumple esta condición también, por lo que hemos comprobado, a través del Teorema del Invariante y del Teorema de Terminación, que el ciclo es correcto y siempre termina. Pero como hay dos instrucciones antes de que comience el ciclo, no hemos terminado. Veamos que, a través de esas dos instrucciones -que llamaremos  $\text{T1}$  y  $\text{T2}$ , y  $\text{T}$  a su conjunto-, se cumple la tripla de Hoare  $[\text{P}] \text{T} [\text{Pc}]$ .

$i\text{P} \longrightarrow_L \text{wp}(\text{T}, \text{Pc})$ ? Calculamos  $\text{wp}(\text{T}, \text{Pc})$ :

$$\text{wp}(\text{T}, \text{Pc}) = \text{wp}(\text{res} := 0, \text{wp}(\text{T2}, \text{Pc})) = \text{wp}(\text{res} := 0; i := 0, \text{P} \wedge i = 0 \wedge \text{res} = 0)$$

Usando dos veces el axioma 1, vemos que  $i = 0$  y  $\text{res} = 0$  son dos predicados *True*, y por lo tanto  $\text{Pc}$  se reduce a:  $(\text{P} \wedge \text{True} \wedge \text{True})$ . Entonces  $\text{wp}(\text{T}, \text{Pc}) = \text{P}$ , y es una tautología decir que  $\text{P} \longrightarrow \text{P}$ .

Queda demostrado, entonces, que la implementación es correcta con respecto de la especificación.

## 2. Demostrar que el valor devuelto es mayor a 50.000.

Ya hemos probado que la implementación es correcta de acuerdo a la especificación. Es decir que  $P \longrightarrow wp(S, Q)$ , siendo  $Q: res = \sum_{i=0}^{|ciudades|-1} ciudades[i].habitantes$ . Ahora, queremos demostrar que, dada cualquier lista que cumpla  $P$ ,  $res > 50.000$ . Sabemos que  $(\exists i : \mathbb{Z}) ((0 \leq i < |ciudades|) \wedge_L (ciudades[i].habitantes > 50,000))$ , por lo que analizaremos distintos casos de la lista *ciudades* teniendo esto en cuenta:

$|ciudades| = 0 \longrightarrow \neg(\exists i : \mathbb{Z}) ((0 \leq i < 0) \wedge_L (ciudades[i].habitantes > 50,000))$ . Por lo tanto,  $|ciudades| > 0$

$|ciudades| = 1 \longrightarrow (\exists i : \mathbb{Z}) ((0 \leq i < 1) \wedge_L (ciudades[i].habitantes > 50,000))$ . Como la lista tiene solo 1 elemento, ese elemento tiene que ser mayor a 50.000, por lo que el resultado también lo será.

$|ciudades| > 1$  : llamaremos **K** a la posición particular donde el valor sea mayor a 50.000, que sabemos que tiene que existir, ya que así lo establece  $P$ . Podemos decir entonces...

$$res = \sum_{i=0}^{k-1} (ciudades[i].habitantes) + ciudades[k].habitantes + \sum_{i=k+1}^{|ciudades|-1} (ciudades[i].habitantes)$$

Notemos que esta igualdad es verdadera, ya que resulta de separar el valor original de  $res$  en distintas series con índices diferentes. Sabemos que, por lo que establecimos anteriormente, que  $ciudades[k].habitantes > 50.000$ . Basta entonces con probar que ambas sumatorias son igual o mayores a 0 para que  $res > 50.000$ .

$$k \neq 0 \wedge k \neq (|ciudades| - 1) \longrightarrow 0 \leq k - 1 < k + 1 \leq |ciudades| - 1.$$

Es decir, ambas sumatorias están en el rango de la lista. La precondition  $P$  también nos dice que...

$$(\forall i : \mathbb{Z}) ((0 \leq i < |ciudades|) \longrightarrow_L (ciudades[i].habitantes \geq 0))$$

Para toda posición dentro de la lista, se cumple que su valor es igual o mayor a 0. Como vimos que ambas sumatorias están, efectivamente, en rango de la lista, entonces cada uno de sus términos valdrá, como mínimo, 0, ¡que era la conclusión a la que queríamos llegar! Nótese lo siguiente:

$$k = 0 \longrightarrow res = \sum_{i=0}^{-1} (ciudades[i].habitantes) + ciudades[0].habitantes + \sum_{i=1}^{|ciudades|-1} (ciudades[i].habitantes)$$

La primera sumatoria, como sus índices están fuera de rango, es igual a 0.  $ciudades[0].habitantes > 50.000$ , y la otra serie será igual o mayor a 0, por lo demostrado anteriormente. La variable *res* será, entonces, mayor a 50.000. Lo mismo sucede si  $k = |ciudades| - 1$ , solo que en ese caso, será la segunda sumatoria, cuyo índice inferior es  $(k + 1)$ , la que se encontrará fuera de rango, pero el razonamiento es el mismo.

Probamos, entonces, que dada cualquier lista que cumpla la especificación, como la implementación del programa es correcta,  $res > 50.000$ .