



# Trabajo Practico de Especificacion

## Especificacion y WP(Weakness Precondition)

9 de septiembre de 2024

Algoritmos y Estructuras de Datos 1

pesutipolimardiano

Integrante	LU	Correo electrónico
Nievas, Martin	453/24	tinnivas@gmail.com
Bercovich, Maximo	002/01	email2@dominio.com
Apellido, Nicolas	003/01	email3@dominio.com
Pomsztein, Andy	624/24	pomszteinandy@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. Enunciados

## 1.1. Especificacion

**1. grandesCiudades:** A partir de una lista de ciudades, devuelve aquellas que tienen más de 50.000 habitantes.  
**proc grandesCiudades** (in ciudades:  $\text{seq}\langle \text{Ciudad} \rangle$ ) :  $\text{seq}\langle \text{Ciudad} \rangle$

**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:  $\text{seq}\langle \text{Ciudad} \rangle$ , in mayoresDeCiudades:  $\text{seq}\langle \text{Ciudad} \rangle$ ) :  $\text{seq}\langle \text{Ciudad} \rangle$

**3. hayCamino:** Un mapa de ciudades está conformado 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  $j$  (Fig. 2). 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.

**proc hayCamino** (in distancias:  $\text{seq}\langle \text{seq}\langle \mathbb{Z} \rangle \rangle$ , in desde:  $\mathbb{Z}$ , in hasta:  $\mathbb{Z}$ ) : Bool

**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  $n$ , ya que indica cuáles pares de nodos poseen 1 camino entre ellos a 1 salto de distancia. Dada la matriz de conexión de orden  $n$ , 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 cantidadNSaltos** (inout conexion:  $\text{seq}\langle \text{seq}\langle \mathbb{Z} \rangle \rangle$ , in n:  $\mathbb{Z}$ )

**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 origen:  $\mathbb{Z}$ , in destino:  $\mathbb{Z}$ , in distancias:  $\text{seq}\langle \text{seq}\langle \mathbb{Z} \rangle \rangle$ ) :  $\text{seq}\langle \mathbb{Z} \rangle$

## 1.2. WP (Weakest Precondition)

La funcion poblacionTotal 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 especificacion:

**proc poblacionTotal** (in ciudades :  $\text{seq}\langle \text{Ciudad} \rangle$ ) :  $\mathbb{Z}$   
 $\text{requiere } \{(\exists i : \mathbb{Z}) (0 \leq i < |\text{ciudades}| \wedge_L \text{ciudades}[i].\text{habitantes} > 50,000 \wedge (\forall i : \mathbb{Z}) (0 \leq i < |\text{ciudades}| \rightarrow_L \text{ciudades}[i].\text{habitantes} \geq 0) \wedge (\forall i, j : \mathbb{Z}) (0 \leq i < j \rightarrow_L \text{ciudades}[i].\text{nombre} \neq \text{ciudades}[j].\text{nombre}))\}$   
 $\text{asegura } \{res = \sum_{i=0}^{|\text{ciudades}|-1} \text{ciudades}[i].\text{habitantes}\}$

Con la siguiente implementacion:

```
1 | res := 0;  
2 | i := 0;  
3 | while (i < ciudades.lenght()) do  
4 |   res := res + ciudades[i].habitantes;  
5 |   i := i + 1  
6 | endwhile
```

- 1. Demostrar que la implementacion es correcta con respecto a la especificacion.
- 2. Demostrar que el valor devuelto es mayor a 50.000.

## 2. Predicados Reutilizables

**pred todosPositivos** (s :  $\text{seq}\langle \mathbb{Z} \rangle$ ) {

```

  (∀i : ℤ) (0 ≤ i < |s| →L s[i] ≥ 0)
}

pred distanciasValidas (distancias : seq⟨seq⟨ℤ⟩⟩) {
  (∀i : ℤ) (0 ≤ i < |distancias| →L todosPositivos(distancias[i]))
}

pred diagonalEnCeros ( s : seq⟨seq⟨ℤ⟩⟩) {
  (∀i, j : ℤ) (0 ≤ i < |s| ∧ 0 ≤ j < s[i] ∧ i = j →L s[i][j] = 0)
}

pred esMatrizSimetrica ( s : seq⟨seq⟨ℤ⟩⟩) {
  (∀i, j : ℤ) (0 ≤ i < |s| ∧ 0 ≤ j < |s[i]| →L s[i][j] = s[j][i])
}

pred esMatrizCuadrada ( s : seq⟨seq⟨ℤ⟩⟩) {
  (∀i : ℤ) (0 ≤ i < |s| →L |s[i]| = |s|)
}

pred esCamino ( distancias : seq⟨seq⟨ℤ⟩⟩, c : seq⟨ℤ⟩, d : ℤ, h : ℤ) {
  (esMatrizCuadrada(distancias) ∧ |c| ≥ 2) ∧L (∀e : ℤ) (e ∈ c →L 0 ≤ e < |distancias| ∧ (c[0] = d ∧ c[|c| - 1] = h) ∧ (∀i : ℤ) (0 ≤ i < |c| - 1 →L distancias[c[i]][c[i + 1]] > 0))
}

```

### 3. Resolucion de Ejercicios

#### 3.1. Especificacion

**Ejercicio 1:**

```

proc grandesCiudades (in ciudades : seq⟨Ciudad⟩) : seq⟨Ciudad⟩
  requiere {true}
  asegura {|res| = cantidadCiudadesGrandes(ciudades) ∧ sonTodasCiudadesGrandes(res) ∧ (∀c : Ciudad) (c ∈ res →L c ∈ ciudades)}

pred sonTodasCiudadesGrandes ( ciudades : seq⟨Ciudad⟩) {
  ((∀i : ℤ) (0 ≤ i < |ciudades| →L ciudades[i].habitantes > 50000))
}

aux cantidadCiudadesGrandes ( ciudades : seq⟨Ciudad⟩) : ℤ = ∑i=0|s|-1 (if s[i].habitantes > 50000 then 1 else 0 fi);

```

**Ejercicio 2:**

```

proc sumaDeHabitantes (in menoresDeCiudades : seq⟨Ciudad⟩, in mayoresDeCiudad : seq⟨Ciudad⟩) : seq⟨Ciudad⟩

  requiere {|menoresDeCiudades| = |mayoresDeCiudades| ∧
    mismasCiudades(menoresDeCiudades, mayoresDeCiudades) ∧
    ciudadesDistintas(menoresDeCiudades) ∧ ciudadesDistintas(mayoresDeCiudades)}

  asegura {|res| = |menoresDeCiudades| ∧ esLaSuma(res, menoresDeCiudades, mayoresDeCiudades) ∧
    mismasCiudades(res, menoresDeCiudades) ∧ mismasCiudades(res, mayoresDeCiudades)}

pred mismasCiudades ( s : seq⟨Ciudad⟩, l : seq⟨Ciudad⟩) {
  (∀i : ℤ) (0 ≤ i < |s| →L (∃j : ℤ) (0 ≤ j < |l| ∧L s[i].nombre = l[j].nombre))
}

pred esLaSuma ( res : seq⟨Ciudad⟩, s : seq⟨Ciudad⟩, l : seq⟨Ciudad⟩) {
  (∀i : ℤ) (0 ≤ i < |res| →L (∃j, k : ℤ) (0 ≤ j < |s| ∧ 0 ≤ k < |l| ∧L s[j].nombre = l[k].nombre ∧
    res[i].habitantes = s[j].habitantes + l[k].habitantes))
}

```

```

}

pred ciudadesDistintas ( ciudades : seq⟨Ciudad⟩ ) {
  (∀i : ℤ) (0 ≤ i < |ciudades| →L ¬(∃j : ℤ) (0 ≤ j < |ciudades| ∧ i ≠ j ∧L ciudades[i].nombre =
ciudades[j].nombre))
}

```

### Ejercicio 3:

```

proc hayCamino (in distancias : seq⟨seq⟨ℤ⟩⟩, in desde : ℤ, in hasta : ℤ) : Bool
  requiere {esMatrizCuadrada(distancias) ∧ diagonalEnCeros(distancias) ∧ esMatrizSimetrica(distancias) ∧
distanciasValidas(distancias) ∧ (0 ≤ desde < |distancias| ∧ 0 ≤ hasta < |distancias|)}
  asegura {res = true ↔ (∃c : seq⟨ℤ⟩) (esCamino(distancias, c, desde, hasta))}

```

### Ejercicio 4:

```

proc cantidadDeCaminosNSaltos (inout conexion : seq⟨seq⟨ℤ⟩⟩, n : ℤ)
  requiere {conexion = conexion0 ∧ esMatrizCuadrada(conexion) ∧ cerosEnLaDiagonal(conexion) ∧
esMatrizSimetrica(conexion) ∧ esMatrizConCerosYUnos(conexion)}
  asegura {|conexion| = |conexion0| ∧L (∀i : ℤ) (0 ≤ i < |conexion0| →L |conexion[i]| = |conexion0| ∧
esMatrizDeOrdenN(conexion, conexion0, n))}

pred esMatrizConCerosYUnos (conexion : seq⟨seq⟨ℤ⟩⟩) {
  (∀i, j : ℤ) (0 ≤ i < |conexion| ∧ 0 ≤ j < |conexion[i]| →L (conexion[i][j] = 0 ∨ conexion[i][j] = 1))
}

pred esIdentidad (m : seq⟨seq⟨ℤ⟩⟩) {
  (esMatrizCuadrada) ∧L (∀i, j : ℤ) (0 ≤ i < |m| ∧ 0 ≤ j < |m[i]| →L ((i = j ∧ m[i][j] = 1) ∨ (i ≠ j ∧ m[i][j] =
0))))
}

pred esProducto (m : seq⟨seq⟨ℤ⟩⟩, n : seq⟨seq⟨ℤ⟩⟩, o : seq⟨seq⟨ℤ⟩⟩, n : ℤ) {
  (∀i, j : ℤ) (0 ≤ i < |m| ∧ 0 ≤ j < |m[i]| →L m[i][j] = ∑k=0|n|-1 n[i][k] * o[k][j])
}

pred esMatrizDeOrdenN (s : seq⟨seq⟨ℤ⟩⟩, l : seq⟨seq⟨ℤ⟩⟩) {
  (∃lista : seq⟨seq⟨seq⟨ℤ⟩⟩⟩) ((|lista| = n + 1 ∧ esIdentidad(lista[0]) ∧ lista[1] = l ∧ lista[n] = s) ∧ (∀i : ℤ) (1 ≤
i ≤ n →L (esProducto(lista[i], lista[i - 1], lista[1]))))
}

```

### Ejercicio 5:

```

proc caminoMinimo (in origen : ℤ, in destino : ℤ, in distancias : seq⟨seq⟨ℤ⟩⟩) : seq⟨ℤ⟩
  requiere {(diagonalEnCeros(distancias) ∧ esMatrizCuadrada(distancias) ∧ esMatrizSimetrica(distancias) ∧
distanciasValidas(distancias) ∧ (0 ≤ origen < |distancias| ∧ 0 ≤ destino < |distancias|))}
  asegura {(esCamino(res) ∧L (∀c : seq⟨ℤ⟩) (esCamino(c) →L distanciaRecorrida(distancias, res) ≥
distanciaRecorrida(distancias, c))) ∨ (res = []) ↔ ¬(∃c : seq⟨ℤ⟩) (esCamino(distancias, c, origen, destino))}

aux distanciaRecorrida (distancias : seq⟨seq⟨ℤ⟩⟩, c : seq⟨ℤ⟩) : ℤ = ∑i=0|c|-2 distancias[c[i]][c[i + 1]];

```

## 4. WP (Weakest Precondition)

### Primer item:

Para demostrar la correctitud de la implementación del programa con respecto a su especificación, debemos ver los siguientes puntos:

- 1. **Precondicion, guarda y postcondicion del ciclo**
- 2.  $P \rightarrow P_c \iff P_c \rightarrow \text{Wp}(\text{res} := 0; \text{Wp}(i = 0, P_c))$  (La precondición de la especificación implica la precondición del ciclo.)

- 3. Correctitud parcial del ciclo
- 4. Terminación del ciclo

#### 4.1. Precondicion, guarda y postcondicion del ciclo

**Precondicion del ciclo:**

$P_c \equiv \{(P \wedge i = 0 \wedge res = 0)\}$  (Donde P es la precondicion de la especificacion).

**Guarda del ciclo:**

$B \equiv \{(i < |ciudades|)\}$

**Postcondicion del ciclo:**

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

Una vez definido la precondicion, guarda y postcondicion ahora veamos si P (la precondicion de la especificacion) implica la  $P_c$  (la precondicion de ciclo) eso lo vamos a hacer calculando la  $Wp(res := 0 ; Wp(i :=, P_c))$

#### 4.2. $P \longrightarrow P_c$

**Cálculo de la  $Wp(res := 0 ; Wp(i := 0, P_c))$ :**

llamemos ① a  $Wp(i := 0, P_c)$  y ②  $Wp(res := 0, ①)$  (utilizo el axioma 3)

**Calculamos ①:**

$Wp(i := 0, P_c) \equiv (def(i) \wedge_L P \wedge 0 = 0 \wedge res = 0) \equiv (P \wedge true \wedge res = 0) \equiv (P \wedge res = 0)$

Una vez calculado ① lo reemplazamos en ② y nos queda  $Wp(res := 0, P \wedge res = 0)$

**Calculamos ②**

$Wp(res := 0, P \wedge res = 0) \equiv (def(res) \wedge_L P \wedge 0 = 0) \equiv (P \wedge true) \equiv P$

Por lo tanto, como  $Wp(res := 0 ; Wp(i := 0, P_c)) \equiv P$ , podemos concluir que  $P \longrightarrow P_c$ . El siguiente paso en la demostración de correctitud consiste en verificar si el ciclo es parcialmente correcto. Para ello, aplicaremos el teorema del invariante, el cual nos permitirá demostrar que una propiedad invariante se mantiene a lo largo de cada iteración del ciclo, garantizando así la correctitud parcial.

#### 4.3. Correctitud parcial del ciclo mediante teorema del invariante

Para demostrar la correctitud parcial del ciclo necesitamos declarar el invariante, el cual explicado en palabras va a consistir en mantener a i en un rango adecuado para evitar la indefinición y que res sea la suma hasta cierto i

**Obs:** el invariante debe valer antes de cada iteración y al finalizar cada iteración.

**Sea I el invariante definido de la siguiente manera:**

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

Los siguientes axiomas son los que se deben corroborar para verificar que el ciclo es parcialmente correcto:

- ①.  $P_c \longrightarrow I$
- ②.  $\{I \wedge B\} S \{I\}$
- ③.  $\{I \wedge \neg B\} \longrightarrow Q_c$

①.  $P_c \longrightarrow I$  (La precondicion del ciclo implica a el invariante)

Entonces tenemos que  $(P \wedge res = 0 \wedge i = 0) \longrightarrow (i \leq i \leq |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].habitantes)$  pero como  $i = 0 \longrightarrow 0 \leq i \leq |ciudades|$ . Además como consecuencia  $res = \sum_{j=0}^{0-1} ciudades[j].habitantes = \sum_{j=0}^{-1} ciudades[j].habitantes = 0$  por lo tanto queda demostrado que  $P_c \longrightarrow I$ .

②.  $\{I \wedge B\} S \{I\} \longleftrightarrow (\{I \wedge B\} \longrightarrow Wp(S_c, I))$

Calculamos el  $Wp(S_c, I)$  (Donde  $S_c$  es el cuerpo del ciclo) por lo tanto por axioma 3 sería equivalente calcular el  $Wp(res := ciudades[i].habitantes, Wp(i := i + 1, I))$ .

Llamamos ④ a  $Wp(i := i + 1, I)$  y ⑤  $Wp(res := res + ciudades[i].habitantes, ④)$

**Cálculo de ④:**

$Wp(i := i + 1, I) \equiv (def(i) \wedge_L (0 \leq i + 1 \leq |ciudades| \wedge res = \sum_{j=0}^{(i+1)-1} ciudades[j].habitantes)) \equiv (0 \leq i + 1 \leq$

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

Una vez calculado  $\textcircled{A}$  remplazamos en  $\textcircled{B}$  y queda de la siguiente manera  $Wp(res := res + ciudades[j].habitantes, 0 \leq i + 1 \leq |ciudades| \wedge res = \sum_{j=0}^i ciudades[j].habitantes)$

**Calculo de  $\textcircled{B}$ :**

$Wp(res := res + ciudades[j].habitantes, 0 \leq i + 1 \leq |ciudades| \wedge res = \sum_{j=0}^i ciudades[j].habitantes) \equiv ((def(i) \wedge def(ciudades) \wedge 0 \leq i < |ciudades|) \wedge_L (0 \leq i + 1 \leq |ciudades| \wedge res + ciudades[i].habitantes = \sum_{j=0}^i ciudades[j].habitantes))$  ahora me fijo que puedo comprimir los rango y restar ciudades[i] a la sumatoria que seria el ultimo termino. Entonces me queda de la siguiente manera:

$$Wp(res := res + ciudades[i].habitantes, \textcircled{A}) \equiv (0 \leq i < |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].habitantes).$$

Ahora veamos si vale que  $\{I \wedge B\} \longrightarrow Wp(S_c, I)$

Donde  $\{I \wedge B\} \equiv (0 \leq i \leq |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].habitantes \wedge i < |ciudades|)$  comprimo el rango de i y me queda lo siguiente  $(0 \leq i < |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].habitantes)$ . Como  $\{I \wedge B\} \equiv Wp(S_c, I)$  entonces vale la implicacion y la tripla de Hoare es valida.