



# Trabajo Practico de Especificacion

## Especificacion y WP(Weakness Precondition)

8 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 m´as de 50.000 habitantes.  
**proc grandesCiudades** (in ciudades:  $seq\langle Ciudad \rangle$ ) :  $seq\langle Ciudad \rangle$

**2. sumaDeHabitantes:** Por cuestiones de planificacion 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:  $seq\langle Ciudad \rangle$ , in mayoresDeCiudades:  $seq\langle Ciudad \rangle$ ) :  $seq\langle Ciudad \rangle$

**3. hayCamino:** Un mapa de ciudades esta 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  $j$  (Fig. 2). Una distancia de 0 equivale a no haber camino entre  $i$  y  $j$ . Notar que la distancia de una ciudad hacia s´ı misma es cero y la distancia entre A y B es la misma que entre B y A.

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

**4. cantidadCaminosNSaltos:** Dentro del contexto de redes informaticas, nos interesa contar la cantidad de “saltos” que realizan los paquetes de datos, donde un salto se define como pasar por un nodo. Ası como definimos la matriz de distancias, podemos definir la matriz de conexi3n entre nodos, donde cada celda  $i, j$  tiene un 1 si hay un unico 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 conexi3n de orden 1, ya que indica cu´ales pares de nodos poseen 1 camino entre ellos a 1 salto de distancia. Dada la matriz de conexi3n de orden 1, este procedimineto debe obtener aquella de orden  $n$  que indica cuantos caminos de  $n$  saltos hay entre los distintos nodos. Notar que la multiplicaci3n de una matriz de conexi3n de orden 1 consigo misma nos da la matriz de conexi3n de orden 2, y ası sucesivamente.<sup>4</sup>

**proc cantidadNSaltos** s (inout conexi3n:  $seq\langle seq\langle \mathbb{Z} \rangle \rangle$ , in n:  $\mathbb{Z}$ )

**5. caminoMinimo:** caminoMinimo: 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 mas corto entre ambas. En caso de no existir un camino, se debe devolver una lista vacıa.

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

## 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 especificaci3n:

**proc poblacionTotal** (in ciudades :  $seq\langle Ciudad \rangle$ ) :  $\mathbb{Z}$

**requiere**  $\{(\exists i : \mathbb{Z}) (0 \leq i < |ciudades| \wedge_L ciudades[i].habitantes > 50,000 \wedge (\forall i : \mathbb{Z}) (0 \leq i < |ciudades| \longrightarrow_L ciudades[i].habitantes \geq 0) \wedge (\forall i, j : \mathbb{Z}) (0 \leq i < j \longrightarrow_L ciudades[i].nombre \neq ciudades[j].nombre))\}$

**asegura**  $\{res = \sum_{i=0}^{|ciudades|-1} ciudades[i].habitantes\}$

Con la siguiente implementaci3n:

```
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 implementaci3n es correcta con respecto a la especificaci3n.
- 2. Demostrar que el valor devuelto es mayor a 50.000.

## 2. Predicados Reutilizables

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

```

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

pred distanciasValidas (distancias : seq⟨seq⟨ $\mathbb{Z}$ ⟩⟩) {
  ( $\forall i : \mathbb{Z}$ ) ( $0 \leq i < |distancias| \longrightarrow_L todosPositivos(distancias[i])$ )
}

pred diagonalEnCeros ( s : seq⟨seq⟨ $\mathbb{Z}$ ⟩⟩) {
  ( $\forall i, j : \mathbb{Z}$ ) ( $0 \leq i < |s| \wedge 0 \leq j < s[i] \wedge i = j \longrightarrow_L s[i][j] = 0$ )
}

pred esMatrizSimetrica ( s : seq⟨seq⟨ $\mathbb{Z}$ ⟩⟩) {
  ( $\forall i, j : \mathbb{Z}$ ) ( $0 \leq i < |s| \wedge 0 \leq j < |s[i]| \longrightarrow_L s[i][j] = s[j][i]$ )
}

pred esMatrizCuadrada ( s : seq⟨seq⟨ $\mathbb{Z}$ ⟩⟩) {
  ( $\forall i : \mathbb{Z}$ ) ( $0 \leq i < |s| \longrightarrow_L |s[i]| = |s|$ )
}

pred esCamino ( distancias : seq⟨seq⟨ $\mathbb{Z}$ ⟩⟩, c : seq⟨ $\mathbb{Z}$ ⟩, d :  $\mathbb{Z}$ , h :  $\mathbb{Z}$ ) {
  ( $esMatrizCuadrada(distancias) \wedge |c| \geq 2 \wedge_L (|c| \geq 1)(\forall e : \mathbb{Z}) (e \in c \longrightarrow_L 0 \leq c < |distancias| \wedge (c[0] = d \wedge c[|c| - 1] = h) \wedge (\forall i : \mathbb{Z}) (0 \leq i < |c| - 1 \longrightarrow_L distancias[c[i]][c[i + 1]] > 0)$ )
}

```

### 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⟩) {
  (( $\forall i : \mathbb{Z}$ ) ( $0 \leq i < |ciudades| \longrightarrow_L ciudades[i].habitantes > 50000$ ))
}

aux cantidadCiudadesGrandes ( ciudades : seq⟨Ciudad⟩) :  $\mathbb{Z} = \sum_{i=0}^{|s|-1}$  (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⟩) {
  ( $\forall i : \mathbb{Z}$ ) ( $0 \leq i < |s| \longrightarrow_L (\exists j : \mathbb{Z}) (0 \leq j < |l| \wedge_L s[i].nombre = l[j].nombre)$ )
}

pred esLaSuma ( res : seq⟨Ciudad⟩, s : seq⟨Ciudad⟩, l : seq⟨Ciudad⟩) {
  ( $\forall i : \mathbb{Z}$ ) ( $0 \leq i < |res| \longrightarrow_L (\exists j, k : \mathbb{Z}) (0 \leq j < |s| \wedge 0 \leq k < |l| \wedge_L s[j].nombre = l[k].nombre \wedge res[i].habitantes = s[j].habitantes + l[k].habitantes)$ )
}

```

```

}

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) ∧ (distancias[desde][hasta] >
    0))}

```

### 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| ∧ i ≤ 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]] ;

```

## 3.2. WP (Weakest Precondition)

### Primer ítem:

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

- **P** →<sub>P<sub>c</sub></sub>: (La precondition de la especificación implica la precondition del ciclo.)
- **Correctitud parcial del ciclo**
- **Terminación del Ciclo**