DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1: Especificación y WP

"En Búsqueda del Camino"

11 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 : $seq\langle \text{String} \times \mathbb{Z} \rangle$) : $seq\langle \text{String} \times \mathbb{Z} \rangle$
  requiere $\{True\}$
  asegura $\{(\forall i : \mathbb{Z})\ ((0 \leq i < |res|) \longrightarrow_L ((res[i] \in ciudades) \wedge_L (res[i]_1 > 50{,}000)))\}$
  asegura $\{(\forall j : \mathbb{Z})\ (((0 \leq j < |ciudades|) \wedge_L (ciudades[j]_1 > 50{,}000)) \longrightarrow_L (ciudades[j] \in res))\}$

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 : $seq\langle \text{String} \times \mathbb{Z} \rangle$, in mayoresDeCiudades : $seq\langle \text{String} \times \mathbb{Z} \rangle$) : $seq\langle \text{String} \times \mathbb{Z} \rangle$
  requiere $\{|menoresDeCiudades| = |mayoresDeCiudades|\}$
  requiere $\{(\forall i : \mathbb{Z})\ ((0 \leq i < |menoresDeCiudades|) \longrightarrow_L (\#Apariciones(menoresDeCiudades[i]_0, menoresDeCiudades)$
  $\#Apariciones(menoresDeCiudades[i]_0, mayoresDeCiudades)))\}$
  asegura $\{|res| = |menoresDeCiudades|\}$
  asegura $\{(\forall j : \mathbb{Z})\ ((0 \leq j < |res|) \longrightarrow_L (\#Apariciones(res[j]_0, res) = \#Apariciones(res[j]_0, menoresDeCiudades)))\}$
  asegura $\{(\forall x, y, z : \mathbb{Z})\ (((0 \leq x < |res|) \wedge (0 \leq y < |res|) \wedge (0 \leq z < |res|) \wedge_L (menoresDeCiudades[x]_0 =$
  $mayoresDeCiudades[y]_0 = res[z]_0)) \longrightarrow_L (res[z]_1 = menoresDeCiudades[x]_1 + mayoresDeCiudades[y]_1))\}$

aux #Apariciones (ciudad : String, ciudades : $seq\langle \text{String} \times \mathbb{Z} \rangle$) : $\mathbb{Z} = \sum_{k=0}^{|ciudades|-1} IfThenElse(ciudad = ciudades[k]_0, 1, 0)$;

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 : $seq\langle seq\langle \mathbb{Z} \rangle \rangle$, in desde : $\mathbb{Z}$, in hasta : $\mathbb{Z}$) : Bool
  requiere $\{0 \leq desde, hasta < |distancias| \wedge (\forall i, j : \mathbb{Z})\ (0 \leq i, j < |distancias| \longrightarrow_L (|distancias| = |distancias[i]| \wedge_L 0 \leq di$
  asegura $\{res = True \Leftrightarrow (\exists s : seq\langle \mathbb{Z} \rangle)\ (esCamino(distancias, desde, hasta, s))\}$

pred hayCaminoDirecto (c1 : $\mathbb{Z}$, c2 : $\mathbb{Z}$, distancias : $seq\langle seq\langle \mathbb{Z} \rangle \rangle$) {
  $distancias[c1][c2] \neq 0$
}

pred esCamino (in distancias : $seq\langle seq\langle \mathbb{Z} \rangle \rangle$, in desde : $\mathbb{Z}$, in hasta: $\mathbb{Z}$, in camino:$seq\langle \mathbb{Z} \rangle$) {
  $|camino| > 1 \wedge_L camino[0] = desde \wedge camino[|camino| - 1] = hasta \wedge 0 \leq hasta < |camino| \wedge ((\forall i : \mathbb{Z})\ (0 \leq i <$
  $|res| - 1 \longrightarrow_L (0 \leq camino[i] < |distancias| \wedge_L hayCaminoDirecto(camino[i], camino[i+1], distancias))))$
}

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

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 procedimineto 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.

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 : $seq\langle seq\langle \mathbb{Z}\rangle\rangle$, in desde : $\mathbb{Z}$, in hasta: $\mathbb{Z}$) : $seq\langle\mathbb{Z}\rangle$
requiere $\{0 \leq desde, hasta < |distancias| \wedge (\forall i, j : \mathbb{Z})\ (0 \leq i, j < |distancias| \longrightarrow_L (|distancias| = |distancias[i]| \wedge_L 0 \leq di$
asegura $\{res = "listavacia" \Leftrightarrow \neg hayCamino(distancias, desde, hasta) \wedge (hayCamino(distancias, desde, hasta) \longrightarrow_L$
$(esCamino(distancias, desde, hasta, res) \wedge_L \neg(\exists s : seq\langle\mathbb{Z}\rangle)\ (esCamino(distancia, desde, hasta, s) \wedge_L distancia(distancias, s$

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 : $seq\langle \text{String} \times \mathbb{Z}\rangle$) : $\mathbb{Z}$
requiere $\{(\exists i : \mathbb{Z})\ ((0 \leq i < |ciudades|) \wedge_L (ciudades[i].habitantes > 50{,}000)) \wedge$
$(\forall i : \mathbb{Z})\ ((0 \leq i < |ciudades|) \longrightarrow_L (ciudades[i].habitantes \geq 0)) \wedge$
$(\forall i, j : \mathbb{Z})\ ((0 \leq i < j < |ciudades|) \longrightarrow_L (ciudades[i].nombre \neq ciudades[j].nombre))\}$
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 $\longrightarrow_L$ wp(S,Q). En primer lugar, intentaremos identificar esta precondición 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: $\mathbf{Q} = \sum_{i=0}^{|ciudades|-1} ciudades[i].habitantes$

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, $\mathbf{Q = Qc}$. 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 precondición del ciclo. Sabemos que esta será la precondición 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 Pc lo será también, sumadas las dos instrucciones antes de llegar al ciclo. Por lo tanto, $\mathbf{Pc = P \wedge i = 0 \wedge res = 0}$

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

1. Pc $\longrightarrow$ I:

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

   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 $\longrightarrow$ Qc

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

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

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

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

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

   wp(S, I) = wp(S1, wp(S2, I)) = wp(res = res + ciudades[i].habitantes, wp(S2, I))

   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:

   wp(S2, I) = 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...

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

   Retomando con lo anterior, obtenemos que:

   wp(S, I) = 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:

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

   Como res no tiene restricciones, esa definición es *True* . Sin embargo, ciudades[i] requiere que i esté en rango de la lista. Por otro lado, podemos decir que $\sum_{j=0}^{i} ciudades[j].habitantes = \sum_{j=0}^{i-1}(ciudades[j].habitantes) + ciudades[i].habitantes$
   Veamos cómo sigue:

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

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

Recapitulando...

($0 \leq i < |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].habitantes$) $\longrightarrow_L$ wp(S, 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 (fv) = $|ciudades| - i$. Veamos si cumple las condiciones:

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

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

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

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

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

2. .[I ∧ B ∧ V0 = fv] S [fv < V0]

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

Usando S, dividido en S1 y S2 como fue explicado anteriormente, podemos decir que:

wp(S, fv < V0) = wp(res = res + ciudades[i].habitantes, wp(S2, fv < V0))

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

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

$(0 \leq i \leq |ciudades| \wedge_L |ciudades| - i - 1 < |ciudades| - i) \longrightarrow fv < V0$. 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 T1 y T2, y T a su conjunto-, se cumple la tripla de Hoare [P] T [Pc].

¿P $\longrightarrow_L$ wp(T, Pc)? Calculamos wp(T, Pc):

wp(T, Pc) = wp(res := 0, wp(T2, Pc)) = wp(res := 0; i := 0, P ∧ i = 0 ∧ res = 0)
Usando dos veces el axioma 1, vemos que i = 0 y res = 0 son dos predicados *True*, y por lo tanto Pc se reduce a:
(P ∧ True ∧ True). Entonces wp(T, Pc) = P, y es una tautología decir que P ⟶ 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 ⟶ 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...

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

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 precondición 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 $= |ciduades| - 1$, 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.

2.1. esto ya estaba

Lo principal: las fórmulas. Se puede poner en una linea, como $x_i = x_{i-1} + x_{i-2}$, o ponerse más grande:

$$\sum_{i=0}^{n} i \tag{2}$$

Y se pueden citar ecuaciones con `\eqref{nombreDeEq}`: (2)
Ejemplo de itemizado:

- Item 1
- Item 2
- Item 3

Ejemplo de enumerado con menor distancia entre items:

1. Item 1
2. Item 2
3. Item 3

Podemos escribir mucho texto. Mucho texto. Mucho texto. Mucho texto. Mucho texto. Mucho texto. Mucho texto. Mucho texto. Mucho texto. Mucho texto. Mucho texto.
Otro párrafo. Otro párrafo. Otro párrafo. Otro párrafo. Otro párrafo. Otro párrafo. Otro párrafo. Otro párrafo. Otro párrafo. Otro párrafo. Otro párrafo. Otro párrafo. Otro párrafo.

Le agregamos una separación entre párrafos. Le agregamos una separación entre párrafos. Le agregamos una separación entre párrafos. Le agregamos una separación entre párrafos. Le agregamos una separación entre párrafos.

La tabla 1 es un ejemplo de cómo se hace una tabla.

Col1	Col2	Col2	Col3
1	6	87837	787
2	7	78	5415
3	545	778	7507
4	545	18744	7560
5	88	788	6344

Tabla 1: Ejemplo de tabla

La figura 2 es un ejemplo de cómo se agrega una imagen.

Figura 1: Ejemplo de figura

(a) Logo de LaTeX

(b) Logo de TeX

Figura 2: Ejemplo para poner dos figuras juntas. Y citarlas por separado a (a) y (b).

```
1 res := 0;
2 i := 0;
3 while (i < s.size()) do
4     res := res + s[i];
5     i := i + 1
6 endwhile
```

Código 1: Ejemplo de código (usando los estilos de la cátedra, ver las macros para más detalles)

Si se pone un label al `lstlisting`, se puede referenciar: Código 1.

2.2. Macros de la cátedra para especificar

`proc nombre` (in paramIn : $\mathbb{N}$, inout paramInout : $seq\langle\mathbb{Z}\rangle$) : tipoRes
`requiere` {$expresionBooleana1$}
`asegura` {$expresionBooleana2$}
`aux auxiliar1` (parametros) : tipoRes = $expresion$;
`pred pred1` (parametros) {
$expresion$
}

`aux auxiliarSuelto` (parametros) : tipoRes = $expresion$;
`pred predSuelto` (parametros) {
$(\forall variable : tipo)\ (algo \longrightarrow_L expresion)$
}
`pred predSuelto` (parametros) {
$(\exists variable : tipo)\ (algo \wedge_L expresion)$
}