Trabajo Practico 1

Especificacion y WP

16 de septiembre de 2024 | Algoritmos y Estructuras de Datos

assertNotaEquals(10)

Integrante	LU	Correo electrónico
Steg, Victoria	1451/21	`vicsteg99@gmail.com`
Lopez, Sabrina	1046/22	`sabrilo999@gmail.com`
Drelewicz, Santiago	286/20	`santidrelewicz2000@gmail.com`
Bravo, Leonardo	217/24	`leo.bravoparedes@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

Una Ciudad se define como una tupla $< nombre, habitantes >$, donde nombre es un string y habitantes es un número entero que representa la cantidad de personas que viven en la ciudad. Cuando nos referimos a las ciudades como puntos en el mapa, las identificaremos a partir de números naturales.

1. **grandesCiudades**: A partir de una lista de ciudades, devuelve aquellas que tienen más de 50.000 habitantes.

   `proc grandesCiudades` (in ciudades : $seq\langle Ciudad\rangle$) : $seq\langle Ciudad\rangle$

   `requiere` $\{True\}$

   `asegura` $\{(\forall i : \mathbb{Z})\ ((0 \leq i < |ciudades| \wedge_L ciudades[i].habitantes > 50000) \longrightarrow_L ciudades[i] \in res) \wedge$
   $(\forall i : \mathbb{Z})\ (0 \leq i < |res| \longrightarrow_L (res[i] \in ciudades \wedge res[i].habitantes > 50000))\}$

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: $seq\langle Ciudad\rangle$, in mayoresDeCiudades: $seq\langle Ciudad\rangle$) : $seq\langle Ciudad\rangle$

   `requiere` $\{|menoresDeCiudades| = |mayoresDeCiudades| \wedge_L$
   $mismosNombres(menoresDeCiudades, mayoresDeCiudades) \wedge$
   $sinCiudadesRepetidas(menoresDeCiudades) \wedge$
   $(\forall ciudad : Ciudad)\ (ciudad \in menoresDeCiudades \longrightarrow ciudad.habitantes \geq 0))\}$

   `asegura` $\{|res| = |menoresDeCiudades| \wedge_L$
   $(\forall i : \mathbb{Z})\ (0 \leq i < |res| \longrightarrow_L (\exists j, k : \mathbb{Z})\ (0 \leq j, k < |res| \wedge_L$
   $(menoresDeCiudades[j].nombre = mayoresDeCiudades[k].nombre = res[i].nombre \wedge$
   $res[i].habitantes = menoresDeCiudades[j].habitantes + mayoresDeCiudades[k].habitantes)))\}$

   `pred mismosNombres` (ciudades1: $seq\langle Ciudad\rangle$, ciudades2: $seq\langle Ciudad\rangle$) {

   $(\forall i : \mathbb{Z})\ (0 \leq i < |ciudades1| \longrightarrow_L$
   $(\exists j : \mathbb{Z})\ (0 \leq j < |ciudades2| \wedge_L ciudades1[i].nombre = ciudades2[j].nombre))$

   $(\forall i : \mathbb{Z})\ (0 \leq i < |ciudades2| \longrightarrow_L$
   $(\exists j : \mathbb{Z})\ (0 \leq j < |ciudades1| \wedge_L ciudades2[i].nombre = ciudades1[j].nombre))$

   }

   `pred sinCiudadesRepetidas` (ciudades: $seq\langle Ciudad\rangle$) {

   $(\forall i : \mathbb{Z})\ (0 \leq i < |ciudades| \longrightarrow_L$
   $(\forall j : \mathbb{Z})\ ((0 \leq i < |ciudades| \wedge i \neq j) \longrightarrow_L ciudades[i].nombre \neq ciudades[j].nombre))$

   }

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` $\{esMatrizCuadrada(distancias) \wedge_L$
   $(esSimetrica(distancias) \wedge$
   $soloCeroEnLaDiagonal(distancias) \wedge$
   $esPositiva(distancias))\}$

   `asegura` $\{res = True \iff (\exists camino : seq\langle\mathbb{Z}\rangle)\ (caminoValido(camino, distancias, desde, hasta))\}$

   `pred esMatrizCuadrada` (matriz: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {

   $(\forall i : \mathbb{Z})\ (|matriz| > 0 \wedge (0 \leq i < |matriz| \longrightarrow_L |matriz[i]| = |matriz|))$

   }

   `pred esSimetrica` (matriz: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {

   $(\forall i, j : \mathbb{Z})\ (0 \leq i, j < |matriz| \longrightarrow_L matriz[i][j] = matriz[j][i])$

   }

pred `soloCeroEnLaDiagonal` (matriz: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {

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

}

pred `esPositiva` (matriz: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {

$(\forall i, j : \mathbb{Z})\ (0 \leq i, j < |matriz| \longrightarrow_L matriz[i][j] >= 0))$

}

pred `caminoValido` (camino: $seq\langle\mathbb{Z}\rangle$, distancias: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$, desde: $\mathbb{Z}$, hasta: $\mathbb{Z}$) {

$((|camino| \geq 2 \wedge_L (camino[0] = desde \wedge camino[|camino| - 1] = hasta)) \wedge$
$sinRepetidos(camino)) \wedge_L$
$(\forall i : \mathbb{Z})\ (0 \leq i < |camino| - 1 \longrightarrow_L hayCaminoDirecto(camino[i], camino[i + 1], distancias)$

}

pred `hayCaminoDirecto` (i: $\mathbb{Z}$, j: $\mathbb{Z}$, distancias: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {

$0 \leq i, j < |distancias| \wedge_L distancias[i][j] > 0$

}

pred `sinRepetidos` (s: $seq\langle\mathbb{Z}\rangle$) {

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

}

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 : $seq\langle seq\langle\mathbb{Z}\rangle\rangle$, in n : $\mathbb{Z}$) : $\mathbb{Z}$

requiere $\{n \geq 1 \wedge conexion = C_1 \wedge$
$esMatrizCuadrada(conexión) \wedge_L$
$(soloCeroEnLaDiagonal(conexión) \wedge$
$soloCerosOUnos(conexión) \wedge$
$esSimetrica(conexión))\}$

asegura $\{conexión = C_n \iff$
$(\exists matrices : seq\langle matriz\mathbb{Z}\rangle)\ ((|matrices| = n \wedge_L (matrices[0] = C_1 \wedge matrices[n - 1] = C_n)) \wedge_L$
$(\forall i : \mathbb{Z})\ (1 \leq i < n \longrightarrow_L (esMatrizCuadrada(matrices[i]) \wedge |C_1| = |matrices[i]|)) \wedge_L$
$(\forall m : \mathbb{Z})\ (1 \leq m < n \longrightarrow_L esMultiplicaciónDeMatrices(matrices[m - 1], C_1, matrices[m]))\}$

pred `soloCerosOUnos` (in conexion : $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {

$res = true \iff (\forall i : \mathbb{Z})((\forall j : \mathbb{Z})\ (0 \leq i, j < |conexion| \longrightarrow_L (conexion[i][j] = 0 \vee conexion[i][j] = 1))$

}

pred `esMultiplicaciónDeMatrices` (A: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$, B: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$, C: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {

$(\forall i, j : \mathbb{Z})\ (0 \leq i, j < |C| \longrightarrow_L C[i][j] = \sum_{k=0}^{|C_1|-1} A[i][k] \times B[k][j])$

}

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 `caminoMínimo` (in origen : $\mathbb{Z}$, in destino : $\mathbb{Z}$, in distancias : $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) : $seq\langle\mathbb{Z}\rangle$

requiere $\{esMatrizCuadrada(distancias) \wedge_L$
$(esSimetrica(distancias) \wedge$
$solo0EnLaDiagonal(distancias) \wedge$
$esPositiva(distancias))\}$

asegura $\{(caminoValido(res, distancias, origen, destino) \wedge_L$
$(\forall camino : seq\langle\mathbb{Z}\rangle)\ (caminoValido(camino, distancias, origen, destino) \longrightarrow_L$

$distanciaRecorrida(distancias, res) \leq distanciaRecorrida(distancias, camino))) \vee$
$(\neg(\exists camino : seq\langle\mathbb{Z}\rangle)\ (caminoValido(camino, distancias, origen, destino)) \wedge res = \langle\rangle)\}$

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

2. Demostraciones de correctitud

Enunciado: 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 : $seq\langle ciudades\rangle$) : $\mathbb{Z}$
`requiere` $\{(\exists i : \mathbb{Z})\ (0 \leq i < |ciudades| \wedge_L ciudades[i].habitantes > 50000) \wedge$
$(\forall i : \mathbb{Z})\ (0 \leq i < |ciudades| \longrightarrow_L ciudades[i].habitantes \geq 0) \wedge$
$(\forall i : \mathbb{Z})(\forall j : \mathbb{Z})\ (0 \leq i < j < |ciudades| \longrightarrow_L ciudades[i].nombre \neq ciudades[j].nombre)\ \}$
`asegura` $\{res = \sum_{i=0}^{|ciudades|-1} = ciudades[i].habitantes\}$

Con la siguiente implementación:

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

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

Antes de comenzar, cabe aclarar que denotaremos ciudades[i].habitantes= ciudades[i].h y ciudades[i].nombres= ciudades[i].n para que sea más fácil de comprender algunas ecuaciones y no se hagan tan extensas.

Para comprobar que un programa que contiene un ciclo **while** es totalmente correcto, se debe calcular el invariante y la función variente del mismo y comprobar cada uno de los siguientes items:

1. $P_c \rightarrow I$
2. $\{I \wedge B\}S\{I\}$ es válida
3. $I \wedge \neg B \rightarrow Q_c$
4. $\{I \wedge B \wedge f_v = v_o\}S\{f_v < v_o\}$ es válida
5. $I \wedge f_v \leq 0 \rightarrow \neg B$

donde P_c y Q_c son la precondición y la postcondición del ciclo, B es la guarda del **while**, I es el invariante y f_v es la función variante.

En este caso, combinando la precondición original y las dos primeras lineas del programa, la precondición del ciclo queda

$$\begin{aligned}P_c \equiv \{&(\exists k : \mathbb{Z})\ (0 \leq k < |ciudades| \wedge_L ciudades[k].h > 50000)\wedge\\ &(\forall n : \mathbb{Z})\ (0 \leq n < |ciudades| \longrightarrow_L ciudades[n].h \geq 0)\wedge\\ &(\forall p : \mathbb{Z})(\forall j : \mathbb{Z})\ (0 \leq p < j < |ciudades| \longrightarrow_L ciudades[p].n \neq ciudades[j].n)\wedge\\ &res = 0 \wedge i = 0\}\end{aligned}$$

Mientras que las poscondición del ciclo, al no haber lineas por debajo del mismo, resulta igual a la postcondicion indicada por el asegura:

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

Sin embargo, además de comprobar que el ciclo es totalmente correcto, comprobaremos que la precondición de la especificación P implica la precondición del ciclo (P_c) con respecto al programa T, siendo este las primeras 2 líneas. Para ello, verificaremos a continuación en el punto 6:

6. $\{P\}T\{P_c\}$ es válida

Lo primero que haremos es calcular el invariante y la función variante. Para ello tomaremos como ejemplo:

$ciudades = \langle (Bs.As, 60{,}000), (Córdoba, 20{,}000), (SantaFe, 30{,}000)\rangle$ y haremos una tabla para ver cómo se comportan las variables i y res en cada iteración del **while**.

Iteración	i	res	Referencia
0	0	0	$\sum_{j=0}^{0-1} ciudades[j].h = 0$
1	1	60.000	$\sum_{j=0}^{1-1} ciudades[j].h = 60{,}000$
2	2	80.000	$\sum_{j=0}^{2-1} ciudades[j].h = 60{,}000 + 20{,}000 = 80{,}000$
3	3	110.000	$\sum_{j=0}^{3-2} ciudades[j].h = 60{,}000 + 20{,}000 + 30{,}000 = 110{,}000$

Tabla 1: Comportamiento de las variables i y res

Por medio de esta tabla podemos ver que i se mueve entre 0 y la longitud de la secuencia (pues, si i tomase valores más grandes se dejaría de cumplir la guarda del while), y que res es una sumatoria que depende de i en cada iteración, por ende, podemos deducir que el invariante es de la siguiente manera:

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

Por otro lado, buscaremos una función tal que cuando llega a 0, deja de cumplirse la guarda, para que así pueda cumplirse el item 5. Vemos que cuando $i = |ciudades|$ el ciclo termina, por ende ese es el valor que tomaremos como referencia.

Queremos que a medida que i crezca, la función se vaya acercando a 0. Si tomamos

$$f_v = |ciudades| - i$$

Cuando $i = |ciudades|$, $f_v = 0$ (cumple con lo que queríamos)

Ahora, con el invariante y la función variante ya calculadas, pasaremos a comprobar los items mencionados antes:

- $P_c \rightarrow I$

Asumiendo que P_c es verdadera, vemos que si en particular $res = 0 \wedge i = 0$, al reemplazar estos en el invariante:

$$\{0 \leq 0 \leq |ciudades| \wedge_L 0 = \sum_{j=0}^{-1} ciudades[j].h = 0\}$$

Ambas son **verdaderas**, por ende la implicación se cumple.

- $\{I \wedge B\}S\{I\}$ **es válida**

  Para ver que esta tupla es válida debemos comprobar que $\{I \wedge B\} \rightarrow wp(S, I)$ es una tautología. Primero calcularemos la precondición más débil de S con respecto a I. Como el programa S está dividido en dos líneas, en este caso las líneas 4 y 5, las llamaremos S1 y S2 respectivamente y aplicaremos los axiomas 3 y 1.

$$wp(\mathbf{S}, I) \equiv wp(\mathbf{S1;S2}, I) \equiv wp(\mathbf{S1}, wp(\mathbf{S2}, I))$$

$$wp(\mathbf{S2}, I) \equiv def(\mathbf{S2}) \wedge_L I^i_{i+1} \equiv def(i+1) \wedge_L (0 \leq i+1 \leq |ciudades| \wedge_L res = \sum_{j=0}^{i} ciudades[j].h)$$

  Como i está bien definida $\rightarrow def(i+1) \equiv True$

$$
\begin{aligned}
wp(\mathbf{S2}, I) &\equiv True \wedge_L \{0 \leq i+1 \leq |ciudades| \wedge_L res = \sum_{j=0}^{i} ciudades[j].h\} \\
&\equiv \{-1 \leq i \leq |ciudades| - 1 \wedge_L res = \sum_{j=0}^{i} ciudades[j].h\} \\
&\equiv \{-1 \leq i < |ciudades| \wedge_L res = \sum_{j=0}^{i} ciudades[j].h\} \\
&\equiv J
\end{aligned}
$$

Ahora calculamos $wp(\mathbf{S1}, J)$

$$
\begin{aligned}
wp(\mathbf{S1}, J) &\equiv def(S1) \wedge_L J^{res}_{res+ciudades[i].h} \\
&\equiv def(res + ciudades[i].h) \wedge_L (-1 \leq i < |ciudades| \wedge_L res + ciudades[i].h = \sum_{j=0}^{i} ciudades[j].h) \\
&\equiv def(res) \wedge def(ciudades[i].h) \wedge_L (-1 \leq i < |ciudades| \wedge_L res + ciudades[i].h = \sum_{j=0}^{i} ciudades[j].h)
\end{aligned}
$$

Como res está bien definida, y para que $ciudades[i].h$ este bien definida i debe estar en rango

$$
wp(\mathbf{S1}, J) \equiv True \wedge 0 \leq i < |ciudades| \wedge_L (-1 \leq i < |ciudades| \wedge_L res + ciudades[j].h = \sum_{j=0}^{i} ciudades[j].h)
$$

Además, vemos que si despejamos res de la sumatoria podemos hacer un cambio de índice

$$
\begin{aligned}
res &= \sum_{j=0}^{i} ciudades[j].h - ciudades[j].h = ciudades[0].h + ... + ciudades[i-1].h + ciudades[j].h - ciudades[j].h \\
&= \sum_{j=0}^{i-1} ciudades[j].h
\end{aligned}
$$

Entonces, se tiene que

$$
wp(\mathbf{S1}, J) \equiv \{0 \leq i < |ciudades| \wedge_L -1 \leq i < |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].h\}
$$

Dado que i debe pertenecer a ambos rangos

$$
wp(\mathbf{S1}, J) \equiv \{0 \leq i < |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].h\}
$$

Retomando, queríamos ver que $I \wedge B \rightarrow wp(S, I)$

$$
\begin{aligned}
I \wedge B &\equiv \{(0 \leq i < |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].h) \wedge i < |ciudades|\} \\
&\equiv \{0 \leq i < |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].h\}
\end{aligned}
$$

Vemos que $I \wedge B \equiv wp(S, I)$, por ende la implicación es **Verdadera**

- $I \wedge \neg B \to Q_c$

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

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

Como $i \leq |ciudades|$ e $i \geq |ciudades|$, por lo tanto $i = |ciudades|$. Reemplazamos en la sumatoria y evaluamos:

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

Si la expresión anterior es verdadera, entonces, $\{res = \sum_{i=0}^{|ciudades|-1} ciudades[i].h\} \equiv Q_c$ es verdadera, y por lo tanto, se cumple la implicación

- $\{I \wedge B \wedge f_v = v_o\}$ **S** $\{f_v < v_o\}$
Vamos a comprobar que $\{I \wedge B \wedge f_v = v_o\} \to wp(S, f_v < v_o)$ es una tautología. Primero calcularemos la precondición más débil:

$$wp(\mathbf{S}, f_v < v_o) \equiv wp(\mathbf{S1;S2}, f_v < v_o) \equiv wp(\mathbf{S1}, wp(\mathbf{S2}, f_v < v_o))$$

$$wp(\mathbf{S2}, f_v < v_o) \equiv \underbrace{def(i+1)}_{True} \wedge_L (f_v < v_o)^i_{i+1}$$

$$wp(\mathbf{S2}, f_v < v_o) \equiv \{|ciudades| - i - 1 < v_o\}$$

$$\equiv J$$

Ahora calculamos $wp(\mathbf{S1}, J)$:

$$\begin{aligned} wp(\mathbf{S1}, J) &\equiv def(res + ciudades[i].h) \wedge_L J^{res}_{res+ciudades[i].h} \\ &\equiv def(res) \wedge def(ciudades[i].h) \wedge_L |ciudades| - i - 1 < v_o \\ &\equiv 0 \leq i < |ciudades| \wedge_L |ciudades| - i - 1 < v_o \end{aligned}$$

Ahora queremos comprobar la implicación:

$$\begin{aligned} \{I \wedge B \wedge f_v = v_o\} &\equiv \{(0 \leq i \leq |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].h) \wedge (i < |ciudades|) \wedge (|ciudades| - i = v_o)\} \\ &\equiv \{(0 \leq i < |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].h) \wedge (|ciudades| - i = v_o)\} \end{aligned}$$

Si $\{I \wedge B \wedge f_v = v_o\}$ es **verdadera** entonces por la primera parte conseguimos $0 \leq i < |ciudades|$ y por la última tenemos que $|ciudades| - i = v_o$, entonces $|ciudades| - i - 1 < v_o \Rightarrow v_o - 1 < v_o \iff -1 < 0$ para todo v_o, por lo tanto, $wp(S, f_v < v_o)$ también es **verdadera**. Así, $\{I \wedge B \wedge f_v = v_o\} S \{f_v < v_o\}$ es válida.

- $I \wedge f_v \leq 0 \rightarrow \neg B$

$$\neg B \equiv \{|ciudades| \leq i\}$$

$$\begin{aligned}\{I \wedge f_v \leq 0\} &\equiv \{(0 \leq i \leq |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].h) \wedge (|ciudades| - i \leq 0)\} \\ &\equiv \{i = |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].h\}\end{aligned}$$

Es fácil observar que si $\{I \wedge f_v \leq 0\}$ es **verdadera**, entonces i es igual a $|ciudades|$, y por lo tanto, se cumple $\neg B$.

- $\{P\}\mathbf{T}\{P_c\}$

Para que sea más comprensible la demostración, llamaremos

$$\begin{aligned}P \equiv \{&(\exists k : \mathbb{Z})\ (0 \leq k < |ciudades| \wedge_L ciudades[k].h > 50000) \wedge \\ &(\forall n : \mathbb{Z})\ (0 \leq n < |ciudades| \longrightarrow_L ciudades[n].h \geq 0) \wedge \\ &(\forall p : \mathbb{Z})(\forall j : \mathbb{Z})\ (0 \leq p < j < |ciudades| \longrightarrow_L ciudades[p].n \neq ciudades[j].n)\}\end{aligned}$$

$$\begin{aligned}P_c \equiv \{&(\exists k : \mathbb{Z})\ (0 \leq k < |ciudades| \wedge_L ciudades[k].h > 50000) \wedge \\ &(\forall n : \mathbb{Z})\ (0 \leq n < |ciudades| \longrightarrow_L ciudades[n].h \geq 0) \wedge \\ &(\forall p : \mathbb{Z})(\forall j : \mathbb{Z})\ (0 \leq p < j < |ciudades| \longrightarrow_L ciudades[p].n \neq ciudades[j].n) \wedge \\ &res = 0 \wedge i = 0\}\end{aligned}$$

Y calculamos $wp(T, P_c)$

$$\begin{aligned}wp(T, P_c) &\equiv wp(res := 0; i := 0, P_c) \\ &\equiv wp(res := 0, wp(i := 0, P_c)) \\ &\equiv wp(res := 0, (\underbrace{def(i := 0)}_{True} \wedge_L P_{c0}^{\ i})) \\ &\equiv wp(res := 0, (P \wedge res = 0 \wedge 0 = 0)) \\ &\equiv \underbrace{def(res := 0)}_{True} \wedge_L (P \wedge res = 0)_0^{res} \\ &\equiv P \wedge 0 = 0 \equiv \mathbf{P}\end{aligned}$$

Vemos que la $wp(T, P_c)$ es exactamente lo mismo que P, entonces podemos decir que $P \rightarrow wp(T, P_c)$ es **verdadero**.

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

Como vimos antes, la implementación es correcta respecto a la especifición por lo que sabemos que si cumplimos la precondición de la especificación, nuestro programa va a cumplir la poscondición de la especificación.Es decir, que logramos demostrar que el programa es correcto y que el ciclo finaliza siempre.

Sea *ciudades*: $seq\langle Ciudad\rangle$ que cumpla con todos los *requiere* de la especificación, entonces como existe un $k : \mathbb{Z}$, $0 \leq k < ciudades.length$ tal que $ciudades[k].habitantes > 50{,}000$, si o si se tiene que dar que $ciudades.length \geq 1$.

En nuestra implementacion comenzamos con $res = 0$ y vamos paso a paso (i por i) sumándole $ciudades[i].habitantes$, que como vimos antes, no solo son todos estos valores positivos por una de las condiciones del requiere, si no que, cuando $i = k$, estaremos sumando un valor mayor que 50000, por lo que res, el valor devuelto, resulta mayor a 50.000.