DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo práctico 1

Especificación y WP

19 de octubre de 2025 — Algoritmos y Estructuras de Datos

BILS

Integrante	LU	Correo electrónico
Picard, Silvano	637/24	s.rene.picard@gmail.com
de Suto Nagy, Bautista	346/24	bdesutonagy@gmail.com
Romero, Ignacio	681/24	ignacion.romero@outlook.es
Avellaneda, Lautaro	41/22	lautaro.avellaneda@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

1.1. grandesCiudades

1.1.1. Consigna de grandesCiudades

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

1.1.2. Solución de grandesCiudades

```
proc grandesCiudades (in ciudades : seq⟨Ciudad⟩) : seq⟨Ciudad⟩
    requiere {sinRepetidos(ciudades)}
    asegura {sinRepetidos(res)}
    asegura {(∀c : Ciudad) (c ∈ res ⟺ c ∈ ciudades ∧ c.habitantes > 50000)}
pred sinRepetidos (s: seq⟨Ciudad⟩) {
    (∀i, j : ℤ) ((0 ≤ i, j < |s| ∧ i ≠ j) ⟶_L s[i].nombre ≠ s[j].nombre))
}
```

Como precondición elegimos que sea necesario que no haya dos entradas para una misma ciudad, así evitamos que haya menos resultados de los que buscamos. Por ejemplo, si tengo una tupla (Cordoba, 10000) y otra (Cordoba, 60000), no podemos determinar cual de las dos tuplas es válida.

1.2. sumaDeHabitantes

1.2.1. Consigna de 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.

1.2.2. Solución de sumaDeHabitantes

```
proc sumaDeHabitantes (in menoresDeCiudades: seq⟨Ciudad⟩, in mayoresDeCiudades: seq⟨Ciudad⟩) : seq⟨Ciudad⟩
    requiere {sinRepetidos(menoresDeCiudades) ∧ sinRepetidos(mayoresDeCiudades) ∧
    nombresIguales(menoresDeCiudades, mayoresDeCiudades)}
    asegura {nombresIguales(res, mayoresDeCiudades) ∧ sinRepetidos(res)}
    asegura {esSuma(res, menoresDeCiudades, mayoresDeCiudades)}
pred esSuma (ciudades1: seq⟨Ciudad⟩, ciudades2: seq⟨Ciudad⟩, ciudades3: seq⟨Ciudad⟩) {
    (∀i : ℤ) (0 ≤ i < |ciudad1| ⟶_L
    (∃j, k : ℤ) ((0 ≤ j < |ciudad2| ∧ 0 ≤ k < |ciudad3|) ∧_L
    (ciudades1[i].nombre = ciudades2[j].nombre = ciudades3[k].nombre∧ciudades1[i].habitantes = ciudades2[j].habitantes+
    ciudades3[k].habitantes)))
}
pred nombresIguales (ciudades1: seq⟨Ciudad⟩, ciudades2: seq⟨Ciudad⟩) {
    (∀i : ℤ) ((0 ≤ i < |ciudades1| ⟶_L (∃j : ℤ) (0 ≤ j < |ciudades2| ∧_L ciudades1[i].nombre = ciudades2[j].nombre)) ∧
    (∀i : ℤ) ((0 ≤ i < |ciudades2| ⟶_L (∃j : ℤ) (0 ≤ j < |ciudades2| ∧_L ciudades1[i].nombre = ciudades2[j].nombre))
}
```

Como precondición pedimos que ninguno de los dos parámetros de entrada tenga dos ciudades con el mismo nombre, ya que sino no tenemos una manera de decidir cual debemos sumar en la respuesta. Luego usamos el predicado nombresIguales que devuelve verdadero si y solo si todos los nombres de la primera lista de ciudades están en la segunda lista y todos los nombres de la segunda están en la primera. Como pedimos que tengan los mismos nombres y que no haya repetidos esto implica que las dos listas tienen el mismo largo.
Como asegura pedimos primero que el resultado tenga la misma estructura que los parámetros de entrada, es decir, que aparezcan las mismas ciudades que en los parámetros de entrada y que no haya nombres repetidos. Con el predicado esSuma aseguramos que cada ciudad de res tenga como cantidad de habitantes la suma de los habitantes para esa ciudad entre las listas menoresDeCiudades y mayoresDeCiudades.

1.3. hayCamino

1.3.1. Consigna de 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.

1.3.2. Solución de hayCamino

proc hayCamino (in distancias: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$, in desde: $\mathbb{Z}$, in hasta: $\mathbb{Z}$) : Bool
requiere $\{esMatrizCuadrada(distancias) \wedge_L$
$(esMatrizSimetrica(distancias) \wedge cerosEnDiagonal(distancias) \wedge sinNegativos(distancias))\}$
asegura $\{res = true \iff existeCamino(distancias, desde, hasta)\}$

pred esMatrizCuadrada (matriz: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {
$(\forall i : \mathbb{Z})\ (0 \leq i < |matriz| \longrightarrow_L |matriz[i]| = |matriz|)$
}
pred esMatrizSimetrica (matriz: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {
$(\forall i, j : \mathbb{Z})\ (0 \leq i, j \leq |matriz| \longrightarrow_L matriz[i][j] = matriz[j][i]))$
}
pred cerosEnDiagonal (matriz: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {
$(\forall i : \mathbb{Z})\ (0 \leq i < |matriz| \longrightarrow_L matriz[i][i] = 0)$
}
pred sinNegativos (matriz: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {
$(\forall i, j : \mathbb{Z})\ (0 \leq i, j < |matriz| \longrightarrow_L matriz[i][j] \geq 0)$
}
pred esCamino (distancias:$seq\langle seq\langle\mathbb{Z}\rangle\rangle$, desde: $\mathbb{Z}$, hasta: $\mathbb{Z}$, camino: $seq\langle\mathbb{Z}\rangle$,) {
$((|camino| > 1) \wedge camino[0] = desde \wedge camino[|camino| - 1] = hasta \wedge$
$(\forall i : \mathbb{Z})\ (0 \leq i < |camino| \longrightarrow_L 0 \leq camino[i] < |distancias|)) \wedge$
$(\forall j : \mathbb{Z})\ (0 \leq j < |camino| - 1 \longrightarrow_L distancias[camino[j]][camino[j + 1]] \neq 0)$
}
pred existeCamino (distancias: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$, desde: $\mathbb{Z}$, hasta: $\mathbb{Z}$) {
$(\exists s : seq\langle\mathbb{Z}\rangle)(esCamino(distancias, desde, hasta, s))$
}

Como precondiciones, para cumplir la consigna, pedimos que la matriz sea cuadrada, simétrica, que su diagonal sea 0 y que no tenga caminos con distancias negativas. Como algunas de las condiciones dependen de que la matriz sea cuadrada, en los predicados asumimos que estas lo son. Para que no se indefina primero chequeamos que sea una matriz cuadrada y con un $\wedge_L$ chequeamos las otras condiciones.
Para determinar si existe un camino, pedimos que exista una secuencia de enteros que dada la matriz de distancias sea camino entre los parámetros de entrada (desde y hasta). Una secuencia de enteros es camino entre 2 ciudades dada una matriz de distancias si: su primer elemento es la ciudad de origen (desde), su ultimo elemento es la ciudad de destino (hasta), todos sus elementos son ciudades (que se encuentran en rango de la matriz) y al tomar cualquiera dos elementos adyacentes (que sus índices sean consecutivos) hay un camino entre esas dos ciudades (indexar la matriz de distancias con ellas no devuelve 0). Además, para que esta condición pueda cumplirse, el tamaño de la secuencia debe ser mayor a uno, para que existan elementos consecutivos en la secuencia.
Una particularidad de esta especificación es que si el parámetro desde o hasta no son ciudades (no están en el rango de la matriz de distancias), entonces res es False.

1.4. cantidadCaminosNSaltos

1.4.1. Consigna de 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

1.4.2. Solución de cantidadCaminosNSaltos

proc cantidadCaminosNSaltos (in conexion: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$, in n: $\mathbb{Z}$) : $seq\langle seq\langle\mathbb{Z}\rangle\rangle$
requiere $\{n > 0\}$
requiere $\{esMatrizCuadrada(conexion) \wedge_L$
$(esMatrizSimetrica(conexion) \wedge esMatrizBinaria(conexion) \wedge cerosEnDiagonal(conexion) \wedge conexion = C_0)\}$

asegura {$|conexion| = |C_0| \land esMatrizCuadrada(conexion)$}
asegura {$(\exists t : seq\langle seq\langle seq\langle \mathbb{Z}\rangle\rangle\rangle)\ (|t| = n + 1 \land t[1] = C_0 \land esListaDePotencias(t) \land conexion = t[n])$}

pred esMatrizBinaria (matriz:$seq\langle seq\langle \mathbb{Z}\rangle\rangle$) {
$(\forall i, j : \mathbb{Z})\ (0 \leq i, j < |matriz| \longrightarrow_L matriz[i][j] = 0 \lor matriz[i][j] = 1)$
}
pred esMatriz (m:$seq\langle seq\langle \mathbb{Z}\rangle\rangle$) {
$(|m| > 0 \land_L |m[0]| > 0) \land (\forall i : \mathbb{Z})\ (0 \leq i < |matriz| \longrightarrow_L |matriz[i]| = |matriz[0]|)$
}
pred esMult (m1 : $seq\langle seq\langle \mathbb{Z}\rangle\rangle$, m2 : $seq\langle seq\langle \mathbb{Z}\rangle\rangle$, m3 : $seq\langle seq\langle \mathbb{Z}\rangle\rangle$) {
$(esMatriz(m1) \land esMatriz(m2) \land esMatriz(m3)) \land_L (|m3| = |m2[0]| \land |m1| = |m2| \land |m1[0]| = |m3[0]|) \land_L$
$(\forall i, j : \mathbb{Z})\ ((0 \leq i < |m1| \land 0 \leq j < |m1[0]|) \longrightarrow_L m1[i][j] = \sum_{k=0}^{|m1|} m2[i][k] * m3[k][j])$
}
pred esListadePotencias (t : $seq\langle seq\langle seq\langle \mathbb{Z}\rangle\rangle\rangle$) {
$|t| > 1 \land_L (\forall i : \mathbb{Z})\ (1 \leq i < |t| \longrightarrow_L esMult(t[i], t[i-1], t[1]))$
}

Como precondiciones pedimos las mismas que en ejercicio 1.3, pero en vez de pedir que no haya caminos negativos requerimos la condición de que sea una matriz binaria, es decir que todos sus valores sean 1 ó 0. Además agregamos la meta variable en el requiere ya que conexion es un parámetro inout y la condición de que n sea mayor que 0, ya que no esta definido que pasa si n es 0 ó negativo.
El primer asegura dice que la matriz conexion mantiene su tamaño, es decir que tiene la misma cantidad de filas que C_0 y es cuadrada.
Para obtener como resultado la matriz de orden n necesitamos que conexion = C_0^n. Para especificar esto utilizamos el predicado esListadePotencias. El predicado esListadePotencias pide que la lista tenga al menos 2 elementos y que todos ellos (excepto el primero) cumplan ser la multiplicación matricial del elemento anterior de la lista por el segundo elemento de la lista, es decir, deben cumplir que t[i] = t[i-1]*t[1] ($\forall i : \mathbb{Z}$ en rango). Para determinar si una matriz es el producto de otras dos usamos el predicado esMult, este pide que todos los parámetros de entrada sean matrices no vacías, que tengan las dimensiones correctas para poder realizar el producto y que se cumpla la definición de multiplicación matricial.
Si una lista de matrices cumple el predicado esListadePotencias, entonces su primer elemento es la matriz identidad y el enésimo elemento es la enésima potencia del segundo elemento ($t[0] = Id, t[1] = a, t[2] = a^2, ..., t[n] = a^n$). Entonces, si pedimos que exista una lista de matrices de n+1 elementos, cuyo segundo elemento sea C_0 y que cumpla esListadePotencias, y si además conexion es igual al enésimo elemento de la lista, tenemos que conexion = C_0^n.

1.5. caminoMinimo

1.5.1. Consigna de 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 más corto entre ambas. En caso de no existir un camino, se debe devolver una lista vacía

1.5.2. Solución de caminoMinimo

proc caminoMinimo (in origen: $\mathbb{Z}$, in destino: $\mathbb{Z}$, in distancias: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$) : $seq\langle \mathbb{Z}\rangle$
requiere {$esMatrizCuadrada(distancias) \land_L$
$(esMatrizSimetrica(distancias) \land cerosEnDiagonal(distancias \land sinNegativos(distancias))$}
asegura {$existeCamino(distancias, origen, destino) \implies (esCamino(distancias, origen, destino, res)$
$\land esCaminoMinimo(res, distancias, origen, destino))$}
asegura {$\neg existeCamino(distancias, origen, destino) \implies res = []$}

aux distancia (camino:$seq\langle \mathbb{Z}\rangle$, dists: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$) : $\mathbb{Z} = \sum_{i=0}^{|camino|-1} dists[i][i+1]$;
pred esCaminoMinimo (camino: $seq\langle \mathbb{Z}\rangle$, distancias: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$, origen: $\mathbb{Z}$, destino: $\mathbb{Z}$) {
$(\forall s : seq\langle \mathbb{Z}\rangle)\ (esCamino(distancias, origen, destino, s) \implies distancia(camino, distancias) \leq distancia(s, distancias))$
}

Como precondiciones utilizamos las mismas que en el ejercicio 1.3, ya que solo hay que asegurarse que la matriz de distancias este bien formada. Para este ejercicio utilizamos varios predicados ya definidos en el ejercicio 1.3
En los aseguras consideramos dos casos. En caso de que exista un camino, res es un camino valido y res es el camino mas corto entre el origen y el destino. Para esto utilizamos el predicado esCaminoMinimo, este dado un camino y la matriz de distancias, devuelve verdadero si para todo camino valido entre el origen y el destino, este tiene una distancia menor o igual a la del camino ingresado. Juntando las condiciones del primer asegura, res debe ser el camino valido, que empieza en el origen y termina en el destino, de menor distancia. En en el segundo asegura tenemos que en caso de que no exista un camino, entonces res es una lista vacía.

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

Con la siguiente implementación:

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

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

Para demostrar la correctitud de la implementación dada respecto de la especificación de poblacionTotal primero deberemos utilizar el Teorema del invariante y luego el Teorema de terminación. Con esto, pasamos a recordar los dos teoremas y luego la serie de pasos que tomaremos para demostrar la correctitud de la implementación.

Sea un ciclo C de la forma **while**(B) **do** $\{S_C\}$ **endwhile**, P_c, Q_c la precondición y postcondición del ciclo respectivamente, B la guarda del ciclo, S el programa dado, fv la función variante del ciclo y v_0 el valor inicial de la anteriormente mencionada, tengo los siguientes teoremas.

Teorema 2.1 (Teorema del Invariante) *Dado un predicado I que cumple:*

1. $P_c \Rightarrow I$
2. $\{I \wedge B\}\, S\, \{Q_c\}$
3. $I \wedge \neg B \Rightarrow Q_c$

entonces podemos afirmar que el ciclo C es parcialmente correcto para la especificación (P_c, Q_c)

Teorema 2.2 (Teorema de Terminación) *Sea* $\mathbb{V}$ *el conjunto de valores que toman las variables del programa. Si existe una función* $fv : \mathbb{V} \to \mathbb{Z}$ *tal que*

1. $\{I \wedge B \wedge fv = v_0\}\, S\, \{fv < v_0\}$
2. $I \wedge fv \leq 0 \Rightarrow \neg B$

entonces la ejecución del ciclo C siempre termina.

Luego, los pasos a seguir serán:

1. $P_c \Rightarrow I$
2. $\{I \wedge B\}\, S\, \{Q_c\}$
3. $I \wedge \neg B \Rightarrow Q_c$
4. $\{I \wedge B \wedge fv = v_0\}\, S\, \{fv < v_0\}$
5. $I \wedge fv \leq 0 \Rightarrow \neg B$

Empecemos definiendo la precondición del ciclo P_c como

$$P_c = \{res = 0 \wedge i = 0\} \tag{1}$$

Lo hacemos en base a las primeras dos lineas de codigo de la implementación dada. Luego, la postcondición Q_c será igual a la postcondición del procedimiento, es decir,

$$Q_c = \left\{ res = \sum_{i=0}^{|ciudades|-1} ciudades[i].habitantes \right\} \tag{2}$$

También, defino B como $i < |ciudades|$, es decir,

$$B = \{i < |ciudades|\} \tag{3}$$

.

Ahora, para determinar la formula del invariante tomaremos un array compuesto unicamente por los segundos elementos de las tuplas de *ciudades*, es decir, compuesto por el numero de habitantes de las tuplas. A este array lo llamamos pruebaInvariante y lo definimos como $pruebaInvariante = [50000, 50001, 50002]$. Ahora, nos armamos una tabla de valores con columnas de nombre iteración, i y res para ver los valores de estas dos ultimas variables en cada iteración:

Iteración	i	res
0	0	0
1	1	50000
2	2	100001
3	3	150003

Tabla 1: Valores de res e i por número de iteración

Podemos ver que el valor de i va desde 0 hasta la longitud de la lista pruebaInvariante, es decir, hasta el 3. Por tanto se puede decir que el rango de i es: $0 \leq i \leq |pruebaInvariante|$.

Sobre el valor de res, vemos que con cada iteración se le suma al valor anterior res el elemento en el indexado en i, escrito de una manera formal vendría a ser: $\sum_{j=0}^{|pruebaInvariante|-1} pruebaInvariante[i]$. La sumatoria llega hasta $|pruebaInvariante| - 1$ porque el indice i empieza en 0, por tanto el primer elemento al que se accederá es, en este caso, 50000, luego 50002 y así sucesivamente. Como i llega hasta la longitud del array pruebaInvariante voy a poder reemplazar $|pruebaInvariante|$ con i. Para no confundir la i del tope de la sumatoria con la de que sirve de punto de inicio de la misma, cambiamos el indice a j. Así, para este ejemplo, el invariante I_p sería:

$$I_p \equiv \left\{ 0 \leq i \leq |pruebaInvariante| \wedge res = \sum_{j=0}^{i-1} pruebaInvariante[j] \right\} \tag{4}$$

Ahora, si lo generalizamos y lo llevamos a nuestro problema inicial, tendremos el siguiente invariante I, que será el que usaremos a lo largo de esta demostración:

$$I \equiv \left\{ 0 \leq i \leq |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].habitantes \right\} \tag{5}$$

Con todo esto, podemos empezar a demostrar a través del Teorema del Invariante la correctitud parcial de esta implementación. Empecemos viendo si se cumple la implicación $P_c \Rightarrow I$, para eso reemplazamos res = 0 e i = 0 en nuestra I:

$$0 \leq 0 \leq |ciudades| \wedge 0 = \sum_{j=0}^{0-1} ciudades[j].habitantes \iff 0 \leq |ciudades| \wedge 0 = 0 \iff 0 \leq |ciudades| \tag{6}$$

Con esto, ya probamos que vale la implicación $P_c \Rightarrow I$. Como segundo paso tenemos que probar que la Tripla de Hoare $\{I \wedge B\}\, S\, \{I\}$ es válida y para esto basta con ver que $\{I \wedge B\} \Rightarrow wp(S, I)$. Antes de calcular $wp(S, I)$ vemos la conjunción $I \wedge B$:

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

En base a la implementación dada defino S1 y S2 como las líneas 4 y 5 respectivamente de la implementación dada (Ver codigo (2)). Es decir,

$$(S1 \equiv res = res + ciudades[i].habitantes) \wedge (S2 \equiv i = i + 1) \tag{7}$$

Así, pasamos a calcular $wp(S1; S2, I)$:

$$\Rightarrow wp(S1; S2, I) \equiv wp(S1, wp(S2, I))$$

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

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

$$\equiv wp(S1, 0 \leq i + 1 < |ciudades| \wedge res = \sum_{j=0}^{i} ciudades[j].habitantes)$$

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

$$\equiv (def(res) \wedge def(i) \wedge def(ciudades)) \wedge_L 0 \leq i < |ciudades|$$

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

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

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

Ya con $wp(S1; S2, I)$ calculada podemos ver si $\{I \wedge B\} \Rightarrow wp(S, I)$:

$$\Rightarrow I \wedge B \equiv 0 \leq i < |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].habitantes \quad \text{Recuerdo la conjunción } I \wedge B$$

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

Podemos ver a simple vista que la Tripla de Hoare es válida ya que la implicación $\{I \wedge B\} \Rightarrow wp(S, I)$ es también válida (de hecho son exactamente iguales).

Para terminar de verificar que la implementación es parcialmente correcta falta ver si vale $I \wedge \neg B \Rightarrow Q_c$. Primero veremos la conjunción del lado izquierdo de la implicación:

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

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

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

Recordando Q_c (2) vemos que la conjunción anterior es exactamente igual a esta postcondición. Luego, la implicación vale y, por tanto, la implementación dada es parcialmente correcta. Así, solo nos falta verificar que el ciclo termina y para eso, como dijimos anteriormente, debemos usar el Teorema de Terminación. Para esto debo definir la función variante fv y para encontrar una que funcione basta con ver que decrezca en cada iteración. Proponemos la función variante

$$fv = |ciudades| - i \tag{8}$$

Para ver que decrezca con cada iteración podemos tomar un ejemplo en que i va de 0 hasta 4 inclusive ($0 \leq i \leq 4$) y ver como cambia de valores fv respecto a i. Veamoslo en una tabla:

Iteración	i	fv
0	0	4
1	1	3
2	2	2
3	3	1
4	4	0

Tabla 2: Valores de fv e i por número de iteración

Podemos notar que el valor de fv decrece a medida que i aumenta en valor, lo que nos da un buen indicio acerca de que sirve como fv.

Pasando ahora a la segunda parte de la demostración, debemos ver si se cumple la Tripla de Hoare $\{I \wedge B \wedge fv = v_0\}\, S\, \{fv < v_0\}$. Para ver si es válida basta con ver que $\{I \wedge B \wedge fv = v_0\} \Rightarrow wp(S, fv < v_0)$. Primero veamos la conjunción $I \wedge B \wedge fv = v_0$:

$$\Rightarrow I \wedge B \wedge fv = v_0 \equiv 0 \leq i \leq |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].habitantes \wedge i < |ciudades| \wedge |ciudades| - i = v_0$$

$$\equiv 0 \leq i < |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].habitantes \wedge |ciudades| - i = v_0$$

Luego, procedemos a calcular $wp(S1; S2, fv < v_0)$:

$$\begin{aligned}
&\Rightarrow wp(S1, wp(S2, fv < v_0)) \\
&\equiv wp(S1, wp(i := i + 1, fv < v_0)) \\
&\equiv wp(S1, def(i) \wedge_L |ciudades| - (i + 1) < v_0) \\
&\equiv wp(res := res + ciudades[i].habitantes, |ciudades| - i - 1 < v_0) \\
&\equiv (def(res) \wedge def(i) \wedge def(ciudades)) \wedge_L 0 \leq i < |ciudades| \wedge_L |ciudades| - i - 1 < v_0 \\
&\equiv 0 \leq i < |ciudades| \wedge_L |ciudades| - i - 1 < v_0
\end{aligned}$$

Ahora, para ver si es válida la implicación mencionada, vemos que los rangos en los que se mueve i son iguales. Además, si reemplazamos el valor de v_0 en el lado derecho de la implicación por el del lado izquierdo tenemos que $|ciudades| - i - 1 < |ciudades| - i$, lo cual vale siempre. Como en la segunda parte de la Tripla de Hoare no se habla de res, entonces es irrelevante para esta implicación. Con lo dicho anteriormente, podemos afirmar que la Tripla de Hoare $\{I \wedge B \wedge fv = v_0\}\, S\, \{fv < v_0\}$ es válida.

Como último paso solo nos queda ver si se cumple la implicación $I \wedge fv \leq 0 \Rightarrow \neg B$. Primero, vemos la conjunción $I \wedge fv \leq 0$:

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

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

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

Esta implicación es válida pues la expresión $i = |ciudades|$ es más fuerte que la negación de la guarda del ciclo ($\neg B$), es decir, $i = |ciudades| \Rightarrow i \geq |ciudades|$ es una tautología. Entonces tenemos que,

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

Finalmente, luego de probar a través del Teorema del invariante y el de terminación, podemos afirmar que la implementación S es correcta respecto de la especificación dada.

2.2. Demostrar que el valor devuelto es mayor a 50.000

Para poder demostrar esto correctamente debemos primero agregar una postcondición a la especificación, así, ésta pasará a ser:

`proc` `poblaciónTotal` (`in` ciudades: $seq\langle Ciudad\rangle$) : $\mathbb{Z}$
`requiere` $\{(\exists i : \mathbb{Z})\ (0 \leq i < |ciudades| \wedge_L ciudades[i].habitantes > 50{,}000) \wedge$
$(\forall i : \mathbb{Z})\ (0 \leq i < |ciudades| \longrightarrow_L ciudades[i].habitantes \geq 0) \wedge$
$(\forall i : \mathbb{Z})\ (\forall j : \mathbb{Z})(0 \leq i < j < |ciudades| \longrightarrow_L ciudades[i].nombre \neq ciudades[j].nombre)\}$
`asegura` $\{res = \sum_{i=0}^{|ciudades|-1} ciudades[i].habitantes\}$
`asegura` $\{res > 50000\}$

La diferencia yace en que hay un asegura más, que es lo que queremos demostrar, y por tanto nuestra postcondicion del ciclo Q_c también cambiará y ahora será:

$$Q_c = \left\{ res = \sum_{j=0}^{|ciudades|-1} ciudades[j].habitantes \wedge res > 50000 \right\} \tag{9}$$

Nuestra precondición P_c seguirá siendo la misma. Como cambia nuestra Q_c entonces tendremos que modificar nuestro invariante, que pasará a ser:

$$\begin{aligned} I \equiv \{0 \leq i \leq |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].habitantes \\ \wedge\ (\exists k : \mathbb{Z})(0 \leq k < |ciudades| \wedge_L ciudades[k].habitantes > 50000) \\ \wedge\ (\forall h : \mathbb{Z})(0 \leq h < |ciudades| \longrightarrow_L ciudades[h].habitantes \geq 0)\} \end{aligned} \tag{10}$$

Para los primeros dos pasos del Teorema del Invariante (ver teorema 2.1) la demostración es la misma que para la sección anterior de este ejercicio (fuera de que se agregan elementos en forma de conjuncion en I pero no terminan cambiando la demostración en sí). El cambio aparece en el tercer paso, es decir cuando queremos ver la implicación $I \wedge \neg B \Rightarrow Q_c$. Recordamos la conjunción del lado izquierdo de la implicación:

$$\begin{aligned} I \wedge \neg B \equiv 0 \leq i \leq |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].habitantes \\ \wedge(\exists k : \mathbb{Z})(0 \leq k < |ciudades| \wedge_L ciudades[k].habitantes > 50000) \\ \wedge(\forall h : \mathbb{Z})(0 \leq h < |ciudades| \longrightarrow_L ciudades[h].habitantes \geq 0 \\ \wedge i \geq 0 \\ \equiv i = |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j].habitantes \\ \wedge(\exists k : \mathbb{Z})(0 \leq k < |ciudades| \wedge_L ciudades[k].habitantes > 50000) \\ \wedge(\forall h : \mathbb{Z})(0 \leq h < |ciudades| \longrightarrow_L ciudades[h].habitantes \geq 0) \\ \equiv res = \sum_{j=0}^{|ciudades|-1} ciudades[j].habitantes \\ \wedge(\exists k : \mathbb{Z})(0 \leq k < |ciudades| \wedge_L ciudades[k].habitantes > 50000) \\ \wedge(\forall h : \mathbb{Z})(0 \leq h < |ciudades| \longrightarrow_L ciudades[h].habitantes \geq 0) \end{aligned}$$

Queremos probar

$$\begin{aligned} [res = \sum_{j=0}^{|ciudades|-1} ciudades[j].habitantes \\ \wedge(\exists k : \mathbb{Z})(0 \leq k < |ciudades| \wedge_L ciudades[k].habitantes > 50000) \\ \wedge(\forall h : \mathbb{Z})(0 \leq h < |ciudades| \longrightarrow_L ciudades[h].habitantes \geq 0)] \Rightarrow res > 50000 \end{aligned}$$

Primero veamos $(\forall h : \mathbb{Z})(0 \leq h < |ciudades| \longrightarrow_L ciudades[h].habitantes \geq 0)$ y como afecta a nuestro res. Expandiendo el cuantificador universal podemos ver que:

$$\begin{aligned} ciudades[0].habitantes \geq 0 \wedge ciudades[1].habitantes \geq 0 \wedge ... \wedge ciudades[|ciudades| - 1].habitantes \geq 0 \\ \text{siendo h cada uno de los indices} \end{aligned}$$

Es decir su suma será,

$$res = \sum_{j=0}^{|ciudades|-1} ciudades[j].habitantes \geq 0 \tag{11}$$

Ahora veamos $(\exists k : \mathbb{Z})(0 \leq k < |ciudades| \wedge_L ciudades[k].habitantes > 50000)$, nos dice que existe al menos un k entero para el cual cuando lo indexamos a ciudades y vemos sus habitantes éste toma un valor mayor a 50000. Podemos reescribir la sumatoria de la siguente manera:

$$res = ciudades[k].habitantes + \sum_{j=0 \wedge j \neq k}^{|ciudades|-1} ciudades[j].habitantes \tag{12}$$

Como $ciudades[k].habitantes > 50000$ y, por lo anteriormente visto, $\sum_{j=0}^{|ciudades|-1} ciudades[j].habitantes \geq 0$ entonces podemos afirmar que:

$$res = \sum_{j=0}^{|ciudades|-1} ciudades[j].habitantes > 50000 \iff res > 50000 \text{ como queriamos probar} \tag{13}$$

Luego, para terminar de demostrar la correctitud de este programa dada esta nueva especificación debemos verificar que el ciclo termina usando el Teorema de Terminación. Realizando la demostración pudimos ver que era igual a la del punto 2.1, para evitar reescribir lo mismo dejamos solo la del 2.1. Con esto, podemos concluir que el programa es correcto respecto a la especificación, que el ciclo termina y además que el resultado es siempre mayor a 50000 como se había pedido.