



Especificación y wp

En búsqueda del camino

4 de octubre de 2024

Algoritmos y estructuras de datos

Grupo datapath

Integrante	LU	Correo electrónico
Andreone, Joaquin	122/24	jandreone06@gmail.com
Comerci, Lucas	818/24	lukicomerci@gmail.com
Luis, Theo	130/23	theoluis44@gmail.com
Zea, Marcos	405/09	zea.marcos@gmail.com



1. Especificaciones

```

proc grandesCiudades (in ciudades : seq<Ciudad>) : seq<Ciudad>
    requiere {sinRepetidos(ciudades) ∧ cantidadHabitantesValida(ciudades)}
    asegura {|res| ≤ |ciudades| ∧ sinRepetidos(res)}
    asegura {(\forall i : Z) (0 ≤ i < |ciudades| ∧ L ciudades[i]₁ ≥ 50000 →L ciudades[i] ∈ res)}
    asegura {(\forall i : Z) (0 ≤ i < |res| →L res[i] ∈ ciudades)}

proc sumaDeHabitantes (in menoresDeCiudades: seq<Ciudad>, in mayoresDeCiudades: seq<Ciudad>) : seq<Ciudad>
    requiere {sinRepetidos(menoresDeCiudades) ∧ sinRepetidos(mayoresDeCiudades) ∧
    mismasCiudades(menoresDeCiudades, mayoresDeCiudades) ∧
    cantidadHabitantesValida(menoresDeCiudades) ∧ cantidadHabitantesValida(mayoresDeCiudades)}
    asegura {|res| = |menoresDeCiudades|}
    asegura {(\forall i, j : Z) (0 ≤ i, j < |menoresDeCiudades| ∧
    menoresDeCiudades[i]₀ = mayoresDeCiudades[j]₀ →L (\exists k : Z) (0 ≤ k < |res| ∧
    (res[k]₀ = menoresDeCiudades[k]₀ ∧ res[k]₁ = menoresDeCiudades[i]₁ + mayoresDeCiudades[j]₁)))}

proc hayCamino (in distancias : seq<seq<Z>>, in desde: Z, in hasta: Z) : Bool
    requiere {0 ≤ desde, hasta < |distancias| ∧ esSimetrica(distancias) ∧ tieneDiagonalNula(distancias)}
    asegura {res = true ↔ (\exists camino : seq<Z>) (esCamino(distancias, desde, hasta, camino))}

proc cantidadCaminosNSaltos (inout conexion : seq<seq<Z>>, in n: Z)
    requiere {esMatrizDeConexion(conexion) ∧ n ≥ 1 ∧ conexion = Conexion₀}
    asegura {mismasDimensiones(conexion, Conexion₀)}
    asegura {(\exists potenciasDeConexion : seq<seq<Z>>) (|potenciasDeConexion| = n ∧
    potenciasDeConexion[0] = Conexion₀ ∧ todasMismasDimensiones(potenciasDeConexion, Conexion₀) ∧
    sonTodasPotenciasDe(potenciasDeConexion, Conexion₀) ∧ conexion = potenciasDeConexion[n - 1])}

proc caminoMinimo (in origen : Z, in destino: Z, in distancias: seq<seq<Z>>) : seq<Z>
    requiere {0 ≤ origen < |distancias| ∧ 0 ≤ destino < |distancias| ∧ esSimetrica(distancias) ∧
    tieneDiagonalNula(distancias)}
    asegura {res = <> ∨ (esCamino(distancias, origen, destino, res) ∧
    (\forall camino : seq<Z>) (esCamino(distancias, origen, destino, camino) →L
    longitudDeCamino(res, distancias) ≤ longitudDeCamino(camino, distancias)))}

```

1.1. Predicados y funciones auxiliares globales

```

pred sinRepetidos (ciudades: seq<Ciudad>) {
    (\forall i : Z) (0 ≤ i < |ciudades| →L (\forall j : Z) (0 ≤ j < |ciudades| ∧ L ciudades[i]₀ = ciudades[j]₀ →L j = i))
}

pred cantidadHabitantesValida (in ciudades: seq<Ciudad>) {
    (\forall i : Z) (0 ≤ i < |ciudades| →L ciudades[i]₁ ≥ 0)
}

pred esSimetrica (matriz:seq<seq<Z>>) {
    (\forall i : Z) (0 ≤ i < |matriz| →L |matriz[i]| = |matriz|) ∧ (\forall i, j : Z) (0 ≤ i, j < |matriz| ∧ L matriz[i][j] = matriz[j][i])
}

pred tieneDiagonalNula (matriz: seq<seq<Z>>) {
    (\forall i, j : Z) (0 ≤ i, j < |matriz| ∧ i = j →L matriz[i][i] = 0)
}

pred mismasCiudades (in ciudades: seq<Ciudad>, in otrasCiudades: seq<Ciudad>) {
    (\forall i : Z) (0 ≤ i < |ciudades| →L (\exists j : Z) (0 ≤ j < |otrasCiudades| ∧ L ciudades[i]₀ = otrasCiudades[j]₀))
}

pred esCamino (distancias: seq<seq<Z>>, desde: Z, hasta: Z, camino: seq<Z>) {
    camino ≠ <> ∧ L desde = camino[0] ∧ hasta = camino[|camino| - 1] ∧ desde ≠ hasta ∧ (\forall i : Z) (0 ≤ i < |camino| →L
    0 ≤ camino[i] < |distancias|) ∧ (\forall i : Z) (0 ≤ i < |camino| - 1 →L distancias[camino[i]][camino[i + 1]] ≠ 0)
}

pred esMatrizDeConexion (matriz: seq<seq<Z>>) {
    esSimetrica(matriz) ∧ tieneDiagonalNula(matriz) ∧ (\forall i, j : Z) (0 ≤ i, j < |matriz| ∧ i ≠ j →L
    (matriz[i][j] = 1 ∨ matriz[i][j] = 0))
}

pred esProductoDe (matrizA: seq<seq<Z>>, matrizB: seq<seq<Z>>, matriz: seq<seq<Z>>) {
    mismasDimensiones(matrizA, matrizB) ∧ mismasDimensiones(matrizA, matriz) ∧
    (\forall i, j : Z) (0 ≤ i, j < |matriz| →L matriz[i][j] = ∑_{k=0}^{|A|-1} matrizA[i][k] · matrizB[k][j])
}

```

```

}
pred mismasDimensiones (matrizA: seq(seq<Z>)), matrizB: seq(seq<Z>)) {
    |matrizA| = |matrizB| ∧ (∀i : Z) (0 ≤ i < |matrizA| →L |matrizA[i]| = |matrizB[i]|)
}
pred todasMismasDimensiones (matrices: seq(seq<seq<Z>>)), matriz: seq(seq<Z>)) {
    (∀i : Z) (0 ≤ i < |matrices| →L mismasDimensiones(matrices[i], matriz))
}
pred sonTodasPotenciasDe (matrices: seq(seq<seq<Z>>)), matriz: seq(seq<Z>)) {
    (∀i : Z) (1 ≤ i < |matrices| → esProductoDe(matriz, matrices[i - 1], matrices[i]))
}
aux longitudDeCamino (camino: seq<Z>, distancias: seq<seq<Z>>) : Z =
|camino|-2
    ∑i=0 distancias[camino[i]][camino[i + 1]];

```

2. Demostraciones de correctitud

La implementación es correcta con respecto a la especificación

```

1 | S1: res := 0;
2 | S2: i := 0;
3 |   while (i < ciudades.length) do
4 |     S3:   res := res + ciudades[i].habitantes
5 |     S4:   i = i + 1
6 |   endwhile

```

Como se ve en la implementacion de la especificacion, el programa consta solo de un ciclo while. Se tienen que cumplir estas condiciones:

1. $P \implies wp(S1; S2; P_c)$
2. $\{P_c\} \text{while } B \text{ do } S_c \text{ endwhile} \{Q_c\}$
3. $Q_c \implies Q$

donde $S_c = S3; S4$. Para demostrar el punto 2, debemos usar el “Teorema del Invariante” y el “Teorema de terminación de un ciclo”. Ambos se cumplen bajo estas condiciones:

- 2.1 $P_c \implies I$
- 2.2 $\{I \wedge B\} S_c \{I\}$
- 2.3 $I \wedge \neg B \implies Q_c$
- 2.4 $\{I \wedge B \wedge v_0 = fv\} S_c \{fv < v_0\}$
- 2.5 $I \wedge fv \leq 0 \implies \neg B$

Basado en la especificacion y la implementacion propongo:

- $P_c \equiv res = 0 \wedge i = 0$
- $B \equiv i < |ciudades|$
- $S_3 \equiv res := res + ciudades[i]$
- $S_4 \equiv i := i + 1$
- $Q_c \equiv res = \sum_{j=0}^{|ciudades|-1} ciudades[j]$
- $I \equiv 0 \leq i \leq |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j]$
- $fv = |ciudades| - i$

1. $P \implies wp(S1; S2; P_c)$

Calculo la $wp(S1; S2; P_c)$:

$$\begin{aligned} wp(S1; S2; P_c) &\equiv wp(S1, wp(S2, P_c)) \equiv wp(S1, wp(i := 0, res = 0 \wedge i = 0)) \equiv wp(S2, res = 0) \\ &\equiv wp(res := 0, res = 0) \\ wp(S1; S2; P_c) &\equiv True \end{aligned}$$

Verifico que $P \implies wp(S1; S2; P_c)$, o sea que $P \implies True$. Pero esto último es trivial, pues es una tautología.

2.1. $P_c \implies I$

Reemplazo:

$$\begin{aligned} res = 0 \wedge i = 0 \implies 0 \leq i \leq |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j]_1 \\ 0 \leq i \leq |ciudades| \equiv 0 \leq 0 \leq |ciudades| \equiv True \\ res = \sum_{j=0}^{i-1} ciudades[j]_1 \equiv 0 = \sum_{j=0}^{0-1} ciudades[j]_1 = 0 \equiv True \end{aligned}$$

Por lo tanto, se cumple $P_c \implies I$

2.2. $\{I \wedge B\}S_c\{I\}$

Para que la tripla de Hoare sea válida, la precondición debe implicar la “Weakest precondition” del código y la postcondición. Es decir:

$$\{I \wedge B\}S_c\{I\} \iff ((I \wedge B) \implies_L wp(S_c, I))$$

$$\begin{aligned} wp(S_c, I) &\equiv wp(S_3, wp(S_4, I)) \equiv wp(S_3, wp(i := i + 1, 0 \leq i \leq |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j]_1)) \\ &\equiv wp(S_3, def(i + 1) \wedge_L 0 \leq i + 1 \leq |ciudades| \wedge res = \sum_{j=0}^{i+1-1} ciudades[j]_1) \\ &\equiv wp(res := res + ciudades[i]_1, 0 \leq i + 1 \leq |ciudades| \wedge res = \sum_{j=0}^i ciudades[j]_1) \\ &\equiv def(res) \wedge_L def(ciudades[i]_1) \wedge 0 \leq i + 1 \leq |ciudades| \wedge res + ciudades[i]_1 = \sum_{j=0}^i ciudades[j]_1 \\ &\equiv 0 \leq i < |ciudades| \wedge_L 0 \leq i + 1 \leq |ciudades| \wedge res + ciudades[i]_1 = \sum_{j=0}^i ciudades[j]_1 \end{aligned}$$

Combino los intervalos de i y resto en ambas partes de la igualdad $ciudades[i]_1$ para transformar el rango de la sumatoria

$$\equiv 0 \leq i < |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j]_1$$

Como $I \wedge B \equiv wp(S_c, I)$, por lo tanto $(I \wedge B) \implies wp(S_c, I)$ que era lo que necesitaba para probar que la tripla de Hoare sea válida.

2.3. $I \wedge \neg B \implies Q_c$

Reemplazo:

$$I \wedge \neg B \equiv 0 \leq i \leq |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j]_1 \wedge i \geq |ciudades|$$

Si $i \geq |ciudades| \wedge i \leq |ciudades|$ entonces $i = |ciudades|$

$$i = |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j]_1 \implies res = \sum_{j=0}^{|ciudades|-1} ciudades[j]_1 \equiv Q_c$$

2.4. $\{I \wedge B \wedge v_0 = fv\} S_c \{fv < v_0\}$

Para que la tripla de Hoare sea válida, la precondición debe implicar la “Weakest precondition” del programa y la post-condición. Es decir:

$$\{I \wedge B \wedge v_0 = fv\} S_c \{fv < v_0\} \iff ((I \wedge B \wedge v_0 = fv) \implies wp(S_c, fv < v_0))$$

Calculo wp:

$$\begin{aligned} wp(S_c, fv < v_0) &\equiv wp(S_3, wp(S_4, fv < v_0)) \equiv wp(S_3, wp(i := i + 1, |ciudades| - i < v_0)) \\ &\equiv wp(res := res + ciudades[i]_1, def(i + 1) \wedge_L |ciudades| - (i + 1) < v_0) \\ &\equiv def(ciudades[i]) \wedge_L |ciudades| - i - 1 < v_0 \\ &\equiv 0 \leq i < |ciudades| \wedge_L |ciudades| - i - 1 < v_0 \end{aligned}$$

Y desarollo la precondición:

$$I \wedge B \wedge v_0 = fv \equiv 0 \leq i \leq |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j]_1 \wedge i < |ciudades| \wedge |ciudades| - i = v_0$$

Ignoro lo que corresponde a res porque no lo necesito para la demostración, y combino los rangos de i :

$$0 \leq i < |ciudades| \wedge |ciudades| - i = v_0$$

Entonces:

$$\begin{aligned} \{I \wedge B \wedge v_0 = fv\} \implies wp(S_c, fv < v_0) \iff \\ 0 \leq i < |ciudades| \wedge |ciudades| - i = v_0 \implies 0 \leq i < |ciudades| \wedge_L |ciudades| - i - 1 < v_0 \iff \\ |ciudades| - i - 1 < |ciudades| - i \iff -1 < 0 \equiv True \end{aligned}$$

Por lo tanto, la tripla de Hoare es válida.

2.5. $I \wedge fv \leq 0 \implies \neg B$

Desarrollo:

$$I \wedge fv \leq 0 \equiv 0 \leq i \leq |ciudades| \wedge res = \sum_{j=0}^{i-1} ciudades[j]_1 \wedge |ciudades| - i \leq 0 \implies \neg B$$

Ignoro lo que corresponde a res porque no lo necesito para la demostración:

$$\begin{aligned} 0 \leq i \leq |ciudades| \wedge |ciudades| - i \leq 0 \implies i \geq |ciudades| \equiv \\ 0 \leq i \leq |ciudades| \wedge |ciudades| \leq i \implies i \geq |ciudades| \equiv \\ i = |ciudades| \implies i \geq |ciudades| \equiv True \end{aligned}$$

3. $Q_c \implies Q$

$$Q_c \equiv res = \sum_{j=0}^{|ciudades|-1} ciudades[j]_1 \implies res = \sum_{i=0}^{|ciudades|-1} ciudades[i]_1 \equiv Q$$

Completamente trivial, se cumple.

El valor devuelto es mayor a 50.000

Como en mi especificación el valor de entrada $ciudades = seq(Ciudad)$ es un *in*, esto me asegura que la lista $ciudades$ no se modifica durante la demostración, con lo cual hace que P sea valido siempre.

Calculo res a partir de la expresión en Q . Como mi P es valido, tomo $ciudades = seq(Ciudad)$ y tengo que existe un j , donde $0 \leq j < |ciudades|$, tal que $ciudades[j]_1 > 50000$. Basta ver que

$$\begin{aligned} res = \sum_{i=0}^{|ciudades|-1} ciudades[i]_1 &= ciudades[j]_1 + \sum_{i=0 \wedge i \neq j}^{|ciudades|-1} ciudades[i]_1 > 50000 + \sum_{i=0 \wedge i \neq j}^{|ciudades|-1} ciudades[i]_1 > 50000 \\ res > 50000 \end{aligned}$$

con lo cual, el valor devuelto es mayor a 50000.