DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo práctico 1: Especificación y WP

"En búsqueda del camino"

12 de septiembre de 2024 — Algoritmos y Estructuras de Datos

Grupo NRO

Integrante	LU	Correo electrónico
Roko, Tomás Esteban	262/23	tomas.e.roko@gmail.com
Nyari, Lisandro Rafael	773/24	lisandronyari@gmail.com
Braslavsky, Lautaro	827/22	lautybras@gmail.com
Quintana, Joaquin Ezequiel	1356/23	joaquin32flores@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

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 $\{(\forall i : \mathbb{Z})\ (0 \leq i < |ciudades| \longrightarrow_L (ciudades[i] \in res \leftrightarrow ciudades[i].habitantes > 50000)) \land$
  $(\forall j : \mathbb{Z})\ (0 \leq j < |res| \longrightarrow_L res[i] \in ciudades)\}$

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 $menoresDeCiudades : seq\langle Ciudad\rangle$, in $mayoresDeCiudades$: $seq\langle Ciudad\rangle$) : $seq\langle Ciudad\rangle$
  requiere $\{sinRepetir(menoresDeCiudades) \land sinRepetir(mayoresDeCiudades) \land$
  $(|menoresDeCiudades| = |mayoresDeCiudades|) \land$
  $(\forall i : \mathbb{N})\ (0 \leq i < |menoresDeCiudades| \longrightarrow_L (\exists j : \mathbb{N})\ (0 \leq j < |mayoresDeCiudades| \land_L menoresDeCiudades[i].nombre =$
  $mayoresDeCiudades[j].nombre))\}$

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

pred sinRepetir (ciudades: $seq\langle Ciudad\rangle$) {
  $(\forall i,j : \mathbb{Z})\ ((0 \leq i < |ciudades| \land i \neq j) \longrightarrow_L ciudades[i].nombre \neq ciudades[j].nombre)$
}

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 $\{matrizCuadradaYSimetrica(distancias) \land 0 \leq desde < |distancias| \land 0 \leq hasta < |distancias|\}$
  asegura $\{res = true \leftrightarrow (\exists s : seq\langle\mathbb{Z}\rangle)\ (|s| > 1 \land s[0] = desde \land s[|s|-1] = hasta \land$
  $(\forall j : \mathbb{Z})\ (0 \leq j < |s| \longrightarrow_L 0 \leq s[j] < |distancias|)) \land_L$
  $(\forall i : \mathbb{Z})\ ((0 < i < |s|) \longrightarrow_L (distancias[s[i]][s[i-1]] \neq 0))\}$

pred matrizCuadradaYSimetrica ($matriz$: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {
  $(\forall i : \mathbb{Z})\ (0 \leq i < |matriz| \longrightarrow_L |matriz[i]| = |matriz|) \land_L$
  $(\forall j,k : \mathbb{Z})\ (0 \leq j,k < |matriz| \longrightarrow_L matriz[i][j] = matriz[j][i])$
}
pred matrizPositiva (matriz: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {
  $(\forall i,j : \mathbb{Z})\ (0 \leq i,j < |matriz| \land_L matriz[i][j] \geq 0)$
}

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 nodo , 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.

`pred esCuadrada` (matriz: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$) {
$(\forall i : \mathbb{Z})\ (0 \leq i < |matriz| \longrightarrow_L |matriz[i]| = |matriz|)$
}
`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 listaDeMatricesCuadradas` (matrices: $seq\langle seq\langle seq\langle \mathbb{Z}\rangle\rangle\rangle$) {
$(\forall i, j : \mathbb{Z})\ (0 \leq i, j < |matrices| \longrightarrow_L (esCuadrada(matrices[i]) \land esCuadrada(matrices[j]) \land |matrices[i]| = |matrices[j]|))$
}
`pred soloUnosyCeros` (matriz: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$) {
$(\forall i, j : \mathbb{Z})\ (0 \leq i, j < |matriz| \longrightarrow_L matriz[i][j] = 0 \lor matriz[i][j] = 1)$
}
`aux productoMatrices` (matriz1: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$, matriz2: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$) : $seq\langle seq\langle \mathbb{Z}\rangle\rangle$ $= (\forall i, j : \mathbb{Z})\ ((0 \leq i < |matriz1|) \land (0 \leq j < |matriz2[0]|) \longrightarrow_L res[i][j] = \sum_{k=0}^{|matriz1[0]|-1} matriz1[i][k] \cdot matriz2[k][j])$;

`proc cantidadCaminosNSaltos` (`inout` $conexión$: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$, `in` n: $\mathbb{Z}$)
`requiere` $\{1 \leq n \land conexion = A0 \land esCuadrada(conexion) \land esSimetrica(conexion) \land soloUnosyCeros(conexion)\}$
`asegura` $\{(\exists s : seq\langle seq\langle seq\langle \mathbb{Z}\rangle\rangle\rangle)\ (|s| = n \land listaDeMatricesCuadradas(s) \land s[0] = A0 \land_L$
$((\forall i : \mathbb{Z})\ (0 < i < |s| \longrightarrow_L s[i] = productoMatrices(s[i-1], A0)) \land_L s[|s|-1] = conexion))\}$

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 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` $\{matrizCuadradaYSimetrica(distancias) \land 0 \leq desde < |distancias| \land 0 \leq hasta < |distancias|\}$
`asegura` $\{esCamino(distancias, res, desde, hasta) \land_L (\forall s : seq\langle \mathbb{Z}\rangle)\ (esCamino(distancias, s, desde, hasta) \longrightarrow_L$
$sumarDistancias(distancias, s) \leq sumarDistancias(distancias, res))\}$

`pred esCamino` (distancias: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$,s: $seq\langle \mathbb{Z}\rangle$, desde: $\mathbb{Z}$, hasta: $\mathbb{Z}$) {
$(|s| > 1 \land s[0] = desde \land s[|s|-1] = hasta \land (\forall j : \mathbb{Z})\ (0 \leq j < |s| \longrightarrow_L 0 \leq s[j] < |distancias|) \land_L$
$(\forall i : \mathbb{Z})\ ((0 < i < |s|) \longrightarrow_L (distancias[s[i]][s[i-1]] \neq 0)))$
}
`aux sumarDistancias` (distancias: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$, s: $seq\langle \mathbb{Z}\rangle$) : $\mathbb{Z}$ $= \sum_{i=0}^{|distancias|-2} distancias[s[i]][s[i+1]]$;

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

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

2.1.

Definiciones:

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

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

Antes del ciclo

2.1.0. $P \implies wp(S_1, P_c) \equiv \{P\}S_1\{P_c\}$

$wp(S_1, P_c) \equiv wp(res = 0, i = 0, res = 0 \wedge i = 0) \equiv$
$wp(res = 0, wp(i = 0, res = 0 \wedge i = 0)) \equiv wp(res = 0, res = 0 \wedge 0 = 0) \equiv$
$0 = 0 \wedge 0 = 0 \equiv True$ ✓

Ciclo while (Teorema de correctitud parcial)

2.1.1. $Pc \implies I$:

$(\exists j : \mathbb{Z})\ (0 \leq j < |ciudades| \wedge ciudades[j].habitantes > 50000) \wedge$
$(\forall j : \mathbb{Z})\ (0 \leq j < |ciudades| \longrightarrow_L ciudades[j].habitantes \geq 0) \wedge$
$(\forall j, k : \mathbb{Z})\ (0 \leq j < k < |ciudades| \longrightarrow_L ciudades[j].nombres \neq ciudades[k].nombre) \wedge$
$(res = 0 \wedge i = 0) \longrightarrow_L (0 \leq i \leq |ciudades| \wedge res = \sum_{h=0}^{i-1} ciudades[h].habitantes)$

$(res = 0, i = 0) : 0 \leq 0 \leq |ciudades| \wedge 0 = \sum_{h=0}^{-1} ciudades[h].habitantes = 0 \equiv True$ ✓

2.1.2. $\{I \wedge B\}S\{I\} \iff \{I \wedge B\} \implies wp(S, \{I\})$

$wp(S, I) \equiv wp\Big(res := res + ciudades[i].habitantes, wp(i = i + 1, I)\Big) \equiv$
$wp\Big(res := res + ciudades[i].habitantes, (0 \leq i + 1 \leq |ciudades| \wedge_L res = \sum_{h=0}^{i} ciudades[h].habitantes)\Big) \equiv$
$def(ciudades[i].habitantes) \wedge_L 0 \leq i + 1 \leq |ciudades| \wedge_L res + ciudades[i].habitantes = \sum_{h=0}^{i} ciudades[h].habitantes \equiv$
$0 \leq i < |ciudades| \wedge res = \sum_{h=0}^{i}(ciudades[h].habitantes) - ciudades[i].habitantes \equiv$
$0 \leq i < |ciudades| \wedge res = \sum_{h=0}^{i-1} ciudades[h].habitantes$

Así: $0 \leq i < |ciudades| \wedge res = \sum_{h=0}^{i-1} ciudades[h].habitantes \implies (0 \leq i < |ciudades| \wedge res = \sum_{h=0}^{i-1} ciudades[h].habitantes$

Es siempre verdadero, y por lo tanto $\{I \wedge B\}S\{I\}$ es una tripla de Hoare válida. ✓

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

$i \geq |ciudades| \wedge 0 \leq i \leq |ciudades| \wedge res = \sum_{h=0}^{i-1} ciudades[h].habitantes \implies res = \sum_{i=0}^{|ciudades|-1} ciudades[i].habitantes \equiv$
$i = |ciudades| \wedge res = \sum_{h=0}^{i-1} ciudades[h].habitantes \implies res = \sum_{i=0}^{|ciudades|-1} ciudades[i].habitantes \equiv$
$(i = |ciudades|) : res = \sum_{h=0}^{|ciudades|-1} ciudades[h].habitantes \implies res = \sum_{i=0}^{|ciudades|-1} ciudades[i].habitantes$

Es siempre verdadero, y por lo tanto $I \wedge \neg B \implies Q_c$ ✓

Finalización del while (Teorema de terminación)

2.1.4. $\{I \wedge B \wedge f_v = v_0\}S\{f_v < v_0\}$

Con $f_v = (|ciudades| - i)$

$\{I \wedge B \wedge f_v = v_0\}S\{f_v < v_0\} \iff \{I \wedge B \wedge f_v = v_0\} \implies wp(S, \{f_v < v_0\})$
$wp(S, \{f_v < v_0\}) \equiv wp\Big(res := res + ciudades[i].habitantes, wp(i := i + 1, (|ciudades| - i) < v_0)\Big) \equiv$
$wp(res + ciudades[i].habitantes, |ciudades| - i - 1 < v_0) \equiv def(ciudades[i].habitantes) \wedge_L |ciudades| - i - 1 < v_0 \equiv$
$0 \leq i < |ciudades| \wedge_L |ciudades| - i - 1 < v_0$

Así, $wp(S, \{f_v < v_0\}) \equiv 0 \leq i < |ciudades| \wedge_L |ciudades| - i - 1 < v_0$

Nos queda ver $\{I \wedge B \wedge f_v = v_0\}$, que ya están calculados:
$\{I \wedge B \wedge f_v = v_o\} \equiv (0 \leq i \leq |ciudades| \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].habitantes) \wedge (i < |ciudades|) \wedge (|ciudades| - i) = v_0$
$\equiv (0 \leq i < |ciudades| \wedge (|ciudades| - i = v_0) \wedge_L res = \sum_{j=0}^{i-1} ciudades[j].habitantes)$

Ahora, verifiquemos si $\{I \wedge B \wedge f_v = v_0\} \implies wp(S, \{f_v < v_0\})$ vale:

$v_0 = (|ciudades| - i)$

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

Si en antecedente es verdadero, en particular vale que:
$(0 \leq i < |ciudades| \wedge_L |ciudades| - i - 1 < v_0) \equiv (|ciudades| - i) - 1 < v_0 \equiv v_0 - 1 < v_0 \equiv 0 < 1 \equiv True$ ✓

2.1.5. $I \wedge f_v \leq 0 \implies \neg B$

$I \wedge f_v \leq 0 \implies \neg B \equiv 0 \leq i \leq |ciudades| \wedge |ciudades| - i \leq 0 \implies i \geq |ciudades|$
$\equiv 0 \leq i \leq |ciudades| \wedge i \geq |ciudades| \implies i \geq |ciudades|$
$\equiv i = |ciudades| \implies i \geq |ciudades|$

Si el antecedente es falso, el consecuente es verdadero.
Si el antecedente es verdadero, es decir $i = |ciudades|$, tenemos que $|ciudades| \geq |ciudades|$ es verdadero.

Luego $I \wedge f_v \leq 0 \implies \neg B$ ✓

2.2.

Macros de la cátedra para especificar(CON ESTAS DE BASE NOSOTROS NOS ARMAMOS NUESTRAS PRED, AUX, PROD)

`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) {
    $(\exists variable : tipo)\ (algo \wedge_L expresion)$
}

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