



Trabajo práctico 1: Especificación y WP

”En búsqueda del camino”

9 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⟨Ciudad⟩) : seq⟨Ciudad⟩
  requiere {True}
  asegura {(∀i : Z) (0 ≤ i < |ciudades| →L (ciudades[i] ∈ res ↔ ciudades[i].habitantes > 50000)) ∧
    (∀j : Z) (0 ≤ j < |res| →L res[j] ∈ 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⟨Ciudad⟩, in mayoresDeCiudades : seq⟨Ciudad⟩) : seq⟨Ciudad⟩
  requiere {sinRepetir(menoresDeCiudades) ∧ sinRepetir(mayoresDeCiudades) ∧
    (|menoresDeCiudades| = |mayoresDeCiudades|) ∧ (∀i : N) (0 ≤ i < |menoresDeCiudades| →L (∃j : N) (0 ≤ j <
    |mayoresDeCiudades| ∧L menoresDeCiudades[i].nombre = mayoresDeCiudades[j].nombre))}

  asegura {sinRepetir(res) ∧ (|res| = |menoresDeCiudades|) ∧ (∀i : Z) ((0 ≤ i < |res|) →L (∃j, k : Z) (0 ≤
  j, k < |res| ∧L res[i].nombre = menoresDeCiudades[j].nombre ∧ res[i].nombre = mayoresDeCiudades[k].nombre ∧
  res[i].habitantes = menoresDeCiudades[i].habitantes + mayoresDeCiudades[k].habitantes))}

pred sinRepetir (ciudades : seq⟨Ciudad⟩) {
  (∀i, j : Z) ((0 ≤ i < |ciudades| ∧ i ≠ j) →L ciudades[i].nombre ≠ 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⟨seq⟨Z⟩⟩, in desde : Z, in hasta : Z) : Bool
  requiere {matrizCuadradaYSimetrica(distancias) ∧ 0 ≤ desde < |distancias| ∧ 0 ≤ hasta < |distancias|}
  asegura {res = true ↔ (∃s : seq⟨Z⟩) (|s| > 1 ∧ s[0] = desde ∧ s[|s| - 1] = hasta ∧
    (∀j : Z) (0 ≤ j < |s| →L 0 ≤ s[j] < |distancias|)) ∧L
    (∀i : Z) ((0 < i < |s|) →L (distancias[s[i]][s[i - 1]] ≠ 0))}

pred matrizCuadradaYSimetrica (matriz : seq⟨seq⟨Z⟩⟩) {
  (∀i : Z) (0 ≤ i < |matriz| →L |matriz[i]| = |matriz|) ∧L
  (∀j, k : Z) (0 ≤ j, k < |matriz| →L matriz[i][j] = matriz[j][i])
}
```

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 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.

```
pred esCuadrada (matriz : seq⟨seq⟨Z⟩⟩) {
  (∀i, j : Z) (0 ≤ i, j < |matriz| →L (|matriz[i]| = |matriz[j]|) ∧ (|matriz| = |matriz[0]|))
}
```

```

}
pred listaDeMatricesCuadradas (matrices: seq<seq<seq<ℤ>>>) {
  (∀i, j : ℤ) (0 ≤ i, j < |matrices| →L (esCuadrada(matrices[i]) ∧ esCuadrada(matrices[j]) ∧ |matrices[i]| = |matrices[j]|))
}
pred soloUnosyCeros (matriz: seq<seq<ℤ>>) {
  (∀i, j : ℤ) (0 ≤ i, j < |matriz| →L matriz[i][j] = 0 ∨ matriz[i][j] = 1)
}
aux productoMatrices (matriz1: seq<seq<ℤ>>, matriz2: seq<seq<ℤ>>) : seq<seq<ℤ>> = (∀i, j : ℤ) ((0 ≤ i < |matriz1|) ∧ (0 ≤ j < |matriz2[0]|) →L res[i][j] = ∑k=0|matriz1[0]|-1 matriz1[i][k] · matriz2[k][j]);

proc cantidadCaminosNSaltos (inout conexión : seq<seq<ℤ>>, in n : ℤ)
  requiere {conexión = A0 ∧ matrizCuadradaYSimetrica(conexión) ∧ soloUnosyCeros(conexión)}
  asegura {(∃s : seq<seq<seq<ℤ>>>) (|s| = n ∧ listaDeMatricesCuadradas(s) ∧ s[0] = A0 ∧L
    ((∀i : ℤ) (0 < i < |s| →L s[i] = productoMatrices(s[i-1], A0)) ∧L s[|s|-1] = conexión))}

```

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 : ℤ, in destino : ℤ, in distancias: seq<seq<ℤ>>) : seq<ℤ>
  requiere {matrizCuadradaYSimetrica(distancias) ∧ 0 ≤ desde < |distancias| ∧ 0 ≤ hasta < |distancias|}
  asegura {esCamino(distancias, res, desde, hasta) ∧L (∀s : seq<ℤ>) (esCamino(distancias, s, desde, hasta) →L
    sumarDistancias(distancias, s) ≤ sumarDistancias(distancias, res))}

pred esCamino (distancias: seq<seq<ℤ>>, s: seq<ℤ>, desde: ℤ, hasta: ℤ) {
  (|s| > 1 ∧ s[0] = desde ∧ s[|s|-1] = hasta ∧ (∀j : ℤ) (0 ≤ j < |s| →L 0 ≤ s[j] < |distancias|) ∧L
  (∀i : ℤ) ((0 < i < |s|) →L (distancias[s[i]][s[i-1]] ≠ 0)))
}
aux sumarDistancias (distancias: seq<seq<ℤ>>, s: seq<ℤ>) : ℤ = ∑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<Ciudad>) : ℤ
  requiere {(∃i : ℤ) (0 ≤ i < |ciudades| ∧ ciudades[i].habitantes > 50,000) ∧
    (∀i : ℤ) (0 ≤ i < |ciudades| →L ciudades[i].habitantes ≥ 0) ∧
    (∀i, j : ℤ) (0 ≤ i < |ciudades| ∧ 0 ≤ j < |ciudades| →L ciudades[i].nombre ≠ ciudades[j].nombre)}
  asegura {res = ∑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.

2.2.

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

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

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