



Trabajo práctico 1

Especificación y WP

11 de septiembre de 2024

Algoritmos y Estructura de Datos

SinGrupo4

Integrante	LU	Correo electrónico
Algañaraz, Franco	001/01	francoarga10@gmail.com
Illescas, Marcos	390/14	marcosillescas90@gmail.com
Bahamonde, Matias	694/21	matubaham@gmail.com
Marión, Ian Pablo	004/01	ianfrodin@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

```
proc grandesCiudades (in ciudades: seq⟨Ciudad⟩) : seq⟨Ciudad⟩
  requiere {True}
  asegura {|res| = CantidadCiudadesMayor50000(ciudades) ∧
    (∀i : ℤ)(0 ≤ i < |res| →L (∃j : ℤ)(0 ≤ j < |ciudades| ∧L (res[i] = ciudades[j] ∧ ciudades[j].habitantes > 50000))))}
aux CantidadCiudadesMayor50000 (in s: seq⟨Ciudad⟩) : ℤ =
  ∑i=0|s|-1 if s[i].habitantes > 50000 then 1 else 0 fi;
```

1.2. sumaDeHabitantes

```
proc sumaDeHabitantes (in menoresDeCiudades: seq⟨Ciudad⟩, in mayoresDeCiudades: seq⟨Ciudad⟩) : seq⟨Ciudad⟩
  requiere {|menoresDeCiudades| = |mayoresDeCiudades| ∧
    mismosNombres(menoresDeCiudades, mayoresDeCiudades)}
  asegura {|res| = |menoresDeCiudades| ∧
    (∀i : ℤ)(0 ≤ i < |res| →L (∃j : ℤ)(∃k : ℤ)((0 ≤ j < |menoresDeCiudad| ∧ 0 ≤ k < |mayoresDeCiudad|) ∧L
    siCoincidenNombresSumoHabitantes(res, menoresDeCiudades, mayoresDeCiudades, i, j, k)))}
pred mismosNombres (in s: seq⟨Ciudad⟩, in t: seq⟨Ciudad⟩) {
  (∀i : ℤ)(0 ≤ i < |s| →L (∃j : ℤ)(0 ≤ j < |t| ∧L s[i].nombre = t[j].nombre))
}
pred siCoincidenNombresSumoHabitantes (in s: seq⟨Ciudad⟩, in t: seq⟨Ciudad⟩, in r: seq⟨Ciudad⟩, in i: ℤ, in j: ℤ, in k: ℤ) {
  (s[i].nombre = t[j].nombre ∧ s[i].nombre = r[k].nombre) → s[i].habitantes = t[j].habitantes + r[k].habitantes
}
```

1.3. hayCamino

```
proc hayCamino (in distancias : seq⟨seq⟨ℤ⟩⟩, in desde : ℤ, in hasta : ℤ) : Bool
  requiere {0 ≤ desde < |distancias| ∧ 0 ≤ hasta < |distancias|}
  asegura {res = true ⇔ existeCamino(distancia, desde, hasta)}
pred existeCamino (in distancias : seq⟨seq⟨ℤ⟩⟩, in desde : ℤ, in hasta : ℤ) {
  (∃camino : seq⟨ℤ⟩) (|camino| ≥ 2 ∧ camino[0] = desde ∧ camino[|camino|-1] = hasta ∧ (∀i : ℤ)(0 ≤ i < |camino|-1 →
  distancias[camino[i]][camino[i+1]] > 0))
}
```

1.4. cantidadCaminosNSaltos

```
proc cantidadCaminosNSaltos (in conexion : seq⟨seq⟨ℤ⟩⟩, in n : ℤ) : seq⟨ℤ⟩
  requiere {n ≥ 1 ∧ conexion = C0}
  asegura {conexion = C0n}
```

1.5. caminoMínimo

```
proc caminoMinimo (in origen : ℕ, in destino : ℤ, in distancias : seq⟨seq⟨ℤ⟩⟩) : seq⟨ℤ⟩
  requiere {0 ≤ origen < |distancias| ∧ 0 ≤ destino < |distancias|}
  asegura {(res = [ ] ⇔ existeCamino(distancias, origen, destino) ∨ origen = destino) ∧ (res ≠ [ ] ⇒
  (∀camino : seq⟨ℤ⟩)(camino[0] = origen ∧ camino[|camino|-1] = destino ∧ sumaDistancias(res, distancias) ≤
  sumaDistancias(camino, distancias)))}
aux sumaDistancia (in camino : seq⟨ℤ⟩, in distancias : seq⟨seq⟨ℤ⟩⟩) : ℤ = ∑i=0|camino|-2 distancias[camino[i]][camino[i+1]];
```

2. Demostraciones de correctitud

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

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