UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2019

28/10/19

Trabajo Práctico de Implementación (TPI)

Morfología Matemática

1. Introducción

El Trabajo Práctico de Implementación consiste en que cada grupo debe implementar en C++ funciones propuestas en el TP de Especificación. Para ello deben seguir la especificación dada por la cátedra en este enunciado, y no la propia que habían realizado en el transcurso del Trabajo Práctico de Especificación.

1.1. Funciones C++

La declaración de las funciones a implementar es la siguiente:

```
bool esImagenValida(const imagen &A);
bool sonPixelesConectados(const imagen &A, pixel p, pixel q, int k);
float devolverPromedioAreas(const imagen &A, int k);
sqPixel calcularContorno(const imagen &A, int k);
void cerrarForma(imagen &A, const imagen &B);
int obtenerRegionConectada(imagen &A, const pixel &semilla);
```

Donde declaramos las siguientes estructuras de datos

```
typedef vector<int> pixel;
typedef vector<pixel> sqPixel;
typedef vector<vector<int>> imagen;
```

2. Consignas

- Implementar todas las funciones que se encuentran en el archivo `ejercicios.h`. Para ello, deberán usar la especificación que se encuentra en la última sección del presente enunciado.
- Extender el conjunto de casos de tests de manera tal de lograr una cobertura de líneas del 100 %. En caso de no poder alcanzarla, explicar el motivo. La cobertura debe estar chequeada con herramientas que se verán en laboratorio de la materia.
- No está permitido el uso de librerías de C++ fuera de las clásicas: **math**, **vector**, **tuple**, las de input-output, etc. Consultar con la cátedra por cualquier librería adicional.

Dentro del archivo que se descarguen desde la página de la materia van a encontrar los siguientes archivos y carpetas:

- `definiciones.h`: Aquí están los renombres mencionados arriba.
- `ejercicios.cpp`: Aquí es donde van a volcar sus implementaciones.
- `ejercicios.h`: *headers* de las funciones que tienen que implementar.
- `auxiliares.cpp` y `auxiliares.h`: Donde es posible volcar funciones auxiliares.
- `main.cpp`: Punto de entrada del programa.
- `tests`: Estos son algunos Tests Suites provistos por la materia. Aquí deben completar con sus propios Tests para lograr la cobertura pedida.
- `lib`: Todo lo necesario para correr Google Tests. Aquí no deben tocar nada.
- `CMakeLists.txt`: Archivo que necesita CLion para la compilación y ejecución del proyecto. **NO** deben sobreescribirlo al importar los fuentes desde CLion. Para ello recomendamos:

1. Lanzar el CLION.
2. Cerrar el proyecto si hubiese uno abierto por *default*: `File->Close Project`
3. En la ventana de Bienvenida de CLION, seleccionar `Open Project`
4. Seleccionar la carpeta del proyecto `src`.
5. Si es necesario, cargar el CMakeList.txt nuevamente mediante `Tools->CMake->Reload CMake Project`
6. No olvidarse descomprimir el GTEST.

Es importante recalcar que la especificación de los ejercicios elaborada por la materia es la guía sobre la que debe basarse el equipo a la hora de implementar los problemas.

3. Entregable

La fecha de entrega del TPI es el **04 de Noviembre de 2019**.

1. Entregar una implementación de las funciones anteriormente descritas que cumplan el comportamiento detallado en la Especificación. El entregable debe estar compuesto por todos los archivos necesarios para leer y ejecutar el proyecto y los casos de test adicionales propuestos por el grupo.
2. El proyecto debe subirse en un archivo comprimido en la solapa Trabajos Prácticos en la tarea SUBIR TPI.
3. **Importante: Utilizar la especificación diseñada para este TP, no la solución del TPE!**.
4. **Importante: Es condición necesaria que la implementación pase todos los casos de tests provistos en el directorio tests. Estos casos sirven de guía para la implementación, existiendo otros TESTS SUITES *secretos* en posesión de la materia que serán usados para la corrección.**

4. Especificación

En esta sección se encuentra la Especificación de los ejercicios a resolver, a partir del enunciado del TPE. La implementación de cada ejercicio DEBE SEGUIR OBLIGATORIAMENTE ESTA ESPECIFICACIÓN.

Todas aquellas auxiliares que no se encuentren definidas inmediatamente después del *proc*, se encuentran en la sección de Predicados y Auxiliares comunes.

4.1. Problemas

1. **proc esImagenValida**(**in** $A : imagen$, **out** $result$: Bool).
   El procedimiento establece si la imagen A cumple:
   - A es matriz válida,
   - A es binaria con valores 0 o 1.

```
proc esImagenValida (in A: imagen, out res: Bool) {
    Pre {True}
    Post {res = true ↔ esImgValida(A)}
}
```

2. **proc sonPixelesConectados**(**in** $A : imagen$, **in** $p : pixel$, **in** $q : pixel$, **in** $k : \mathbb{Z}$, **out** $result$: Bool).
   El procedimiento devuelve verdadero si se verifica que existe un camino de píxeles activados de adyacencia k que conecta a p y q en la imagen válida de entrada A.

```
proc sonPixelesConectados (in A: imagen, in p: pixel, in q: pixel, in k:Z, out res: Bool) {
    Pre {esImgValida(A) ∧_L pixelValido(p, A) ∧ pixelValido(q, A) ∧ adyacenciaValida(k)}
    Post {res = true ↔ estanConectados(A, p, q, k)}
}
```

3. **proc devolverPromedioAreas**(**in** $A : imagen$, **in** $k : \mathbb{Z}$, **out** $prom : \mathbb{R}$).
   El procedimiento devuelve el promedio del área de todas las regiones de adyacencia k en la imagen de entrada A. El área la definimos como la cantidad de píxeles que componen una región. En caso de ser una imagen vacia, el promedio es cero.

```
proc devolverPromedioAreas (in A: imagen, in k: ℤ, out prom: ℝ) {
    Pre {esImgValida(A) ∧ adyacenciaValida(k)}
    Post {(hayDato(A) ∧_L esPromedioAreas(A, k, prom)) ∨ (imagenApagada(A) ∧ prom = 0)}
}
pred esPromedioAreas (A: imagen, k: ℤ, val: ℝ) {
    (∃ lR : seq⟨sqPixel⟩) (esListaRegiones(lR, A, k) ∧ val = Σ_{i=0}^{|lR|-1} |lR[i]| / |lR|)
}
```

4. **proc calcularContorno**(**in** $A : imagen$, **in** $k : \mathbb{Z}$, **out** $edge : seq\langle pixel\rangle$).
El procedimiento devuelve el conjunto de píxeles que corresponden al contorno de la forma presente en la imagen de entrada A. En A hay como mínimo una región u objeto con, al menos, 2 pixeles.

```
proc calcularContorno (in A: imagen, in k: ℤ, out edge: sqPixel) {
    Pre {esImgValida(A) ∧_L minimoUnaRegionValida(A) ∧ adyacenciaValida(k)}
    Post {sonTodosLosDelContorno(A, edge, k) ∧ todosPixelesDiferentes(edge)}
}
pred minimoUnaRegionValida (A: imagen, k: ℤ) {
    (∃ lR : seq⟨sqPixel⟩)(esListaRegiones(lR, A, k) ∧ |lR| ≥ 1 ∧ (∃ sR : sqPixel)(sR ∈ lR ∧ |sR| ≥ 2))
}
pred sonTodosLosDelContorno (A: imagen, edge: sqPixel, k: ℤ) {
    (∀ p : pixel)(p ∈ edge ↔ pixelValidoEncendido(p, A) ∧_L (tocaConBackground(p, A, k) ∨ estaEnBorde(p, A)))
}
pred tocaConBackground (p: pixel, A: imagen, k: ℤ) {
    (∃ q : pixel)(pixelValido(q, A) ∧_L ¬activado(q, A) ∧ esAdyacente(p, q, k))
}
pred estaEnBorde (p: pixel, A: imagen) {
    p[0] = 0 ∨ p[0] = |A| − 1 ∨ p[1] = 0 ∨ p[1] = |A[0]| − 1
}
pred todosPixelesDiferentes (pixeles: sqPixel) {
    (∀i : ℤ)(∀j : ℤ)(0 ≤ i < |pixeles| ∧ 0 ≤ j < |pixeles| ∧ i ≠ j ⟶_L pixeles[i] ≠ pixeles[j])
}
```

5. **proc cerrarForma**(**in** $A : imagen$, **in** $b : imagen$).
Aplicar la operación *closing* a la matriz A, no vacía, usando el elemento estructurante b.

```
proc cerrarForma (inout A: imagen, in B: imagen) {
    Pre {esImgValida(A) ∧ esElementoEstructuranteValido(B) ∧_L |A| > |B| ∧ |A[0]| > |B[0]| ∧ A = A_0}
    Post {(∃ D : imagen)(esDilatacion(A_0, B, D) ∧ esErosion(D, B, A))}
}
pred esErosion (A: imagen, B: imagen, E: imagen) {
    esImgValida(E) ∧ mismasDimensiones(A, E) ∧ (∃ sqA : sqPixel) (
        cumpleDualidadImagenSecuencia(sqA, A) ∧ sonTodosLosCentrosDeDiscosContenidos(sqA, B, E)
    )
}
pred sonTodosLosCentrosDeDiscosContenidos (sqA: sqPixel, B: imagen, E: imagen) {
    (∀ z : pixel) (esCoordenada(z) ∧_L pixelEnRango(z, E) ⟶_L (
        activado(z, E) ↔ elementoDesplazadoContenidoEnImagen(B, sqA, z)
    ))
}
```

```
pred elementoDesplazadoContenidoEnImagen (B: imagen, sqIn: sqPixel, z: pixel) {
    (∃ sBp : sqPixel) (esSeqBDesplazada(B, sBp, z) ∧ estaContenida(sBp, sqIn))
}
```

6. **proc obtenerRegionConectada**(**inout** $A : imagen$, **in** $semilla : pixel$, **out** $ite : \mathbb{Z}$).
Implementar el algoritmo de Región Conectada, modificando la imagen de entrada de manera de conservar solo la región conectada a la semilla dada como parámetro. La función devuelve además el número de iteraciones necesarias para convergencia. Tomando el ejemplo de la figura 9 del enunciado del TPE, se calculan 7 matrices X_i, ya que las últimas dos son iguales y representa la condición de fin de ciclo. Sin embargo, para este TPI, la especificación indica que hay que devolver el número de iteraciones que resultaron en una matriz única. De este modo, el valor de $ite = 6$.

```
proc obtenerRegionConectada (inout A : imagen, in semilla : pixel, out ite: ℤ)  {
    Pre {esImgValida(A) ∧ pixelValidoEncendido(semilla, A) ∧ A = A_0}
    Post {(∃ sqI : seq⟨imagen⟩)(secuenciaImagenesValida(sqI, A_0) ∧_L
    esSecuenciaAlgoRegConectada(A_0, A, semilla, sqI) ∧ ite = |sqI|)}
}
pred secuenciaImagenesValida (sqI : seq⟨imagen⟩, A: imagen) {
    |sqI| > 0 ∧ todasValidas(sqI) ∧_L todasImagenesDistintas(sqI) ∧ todasMismasDimensiones(sqI, A)
}
pred todasValidas (sqI : seq⟨imagen⟩) {
    (∀i : ℤ)(0 ≤ i < |sqI| ⟶_L esImgValida(sqI[i]))
}
pred todasMismasDimensiones (sqI : seq⟨imagen⟩, A: imagen) {
    (∀i : ℤ)(0 ≤ i < |sqI| ∧_L esImgValida(sqI[i]) ⟶_L mismasDimensiones(sqI[i], A))
}
pred esSecuenciaAlgoRegConectada (A_0: imagen, A: imagen, semilla: pixel, sqI : seq⟨imagen⟩) {
    soloActivadaSemilla(semilla, sqI[0]) ∧ imagenesSonDilatacionesEIntersecciones(A_0, sqI) ∧
    esLaRegionCompleta(sqI[|sqI| − 1], A_0) ∧  A = sqI[|sqI| − 1]
}
pred soloActivadaSemilla (s: pixel, A: imagen) {
    activado(s, A) ∧ (∀p : pixel)(pixelValido(p, A) ∧ p ≠ s  ⟶_L  ¬activado(A, p))
}
pred todasImagenesDistintas (sqI:seq⟨imagen⟩) {
    (∀i : ℤ)(0 ≤ i < |sqI| − 1 ⟶_L sqI[i] ≠ sqI[i + 1])
}
pred imagenesSonDilatacionesEIntersecciones (A:imagen, sqI:seq⟨imagen⟩) {
    (∀i : ℤ)((0 ≤ i < |sqI| − 1)  ⟶_L  esDilatacionLuegoInterseccion(sqI[i], sqI[i + 1], A))
}
pred esDilatacionLuegoInterseccion (I1:imagen, I2:imagen, A:imagen) {
    (∃ B : imagen)(∃ D : imagen)(esImgValida(D) ∧ mismasDimensiones(D, A) ∧
    esElementoEstructuranteValidoK8(B) ∧ |B| = 3 ∧_L esDilatacion(I1, B, D) ∧ esLaInterseccion(D, A, I2))
}
pred esLaInterseccion (I1: imagen, I2: imagen, Iout: imagen) {
    (∃ sI : sqPixel)(cumpleDualidadImagenSecuencia(sI, Iout) ∧_L
    (∀ p : pixel)(p ∈ sI ↔ pixelValido(p, Iout) ∧_L activado(p, I1) ∧ activado(p, I2))
}
pred esLaRegionCompleta (A:imagen,B:imagen) {
    esDilatacionLuegoInterseccion(A, A, B)
}
```

4.2. Predicados y Auxiliares generales

4.2.1. Usados desde el ejercicio 1

`pred esImgValida` (A: imagen) {
$\neg vacia(A) \wedge_L esMatriz(A) \wedge_L esBinaria(A)$
}
`pred vacia` (A:imagen) {
$|A| = 0$
}
`pred esMatriz` (A:imagen) {
$(\forall i : \mathbb{Z})\ (0 \leq i < |A| \longrightarrow_L |A[i]| \neq 0) \wedge_L (\forall i, j : \mathbb{Z})\ (0 \leq i < j < |A| \longrightarrow_L |A[i]| = |A[j]|)$
}
`pred esBinaria` (A:imagen) {
$(\forall i, j : \mathbb{Z})\ 0 \leq i < |A| \wedge 0 \leq j < |A[0]| \longrightarrow_L A[i][j] = 0 \vee A[i][j] = 1$
}

4.2.2. Usados desde ejercicio 2

`pred adyacenciaValida` (k :$\mathbb{Z}$) {
$k = 4\ \vee\ k = 8$
}
`pred activado` (p: pixel, A: imagen) {
$A[p[0]][p[1]] = 1$
}
`pred esCoordenada` (p: pixel) {
$|p| = 2$
}
`pred pixelEnRango` (p: pixel, A:imagen) {
$p[0] \geq 0 \wedge p[1] \geq 0 \wedge p[0] < |A| \wedge p[1] < |A[0]|$
}
`pred pixelValido` (p: pixel, A: imagen) {
$esCoordenada(p) \wedge_L pixelEnRango(p, A)$
}
`pred pixelValidoEncendido` (p: pixel, A: imagen) {
$pixelValido(p, A) \wedge_L activado(p, A)$
}
`pred cumpleDualidadImagenSecuencia` (sq: sqPixel, A: imagen) {
$(\forall p : pixel)\ (p \in sq \leftrightarrow pixelValidoEncendido(p, A))$
}
`pred estanConectados` (A: imagen, p: pixel, q: pixel, k: $\mathbb{Z}$) {
$(p = q) \vee ((\exists\ camino : sqPixel)(esSecuenciaValida(camino, A) \wedge |camino| \geq 1 \wedge_L$
$camino[0] = p \wedge camino[|camino| - 1] = q \wedge esCaminoConectado(camino, A, k)))$
}
`pred esSecuenciaValida` (camino: sqPixel, A: imagen) {
$|camino| > 0 \wedge ((\forall i : \mathbb{Z})(0 \leq i < |camino| \longrightarrow_L pixelValido(camino[i], A))$
}
`pred esCaminoConectado` (camino: sqPixel, A: imagen, k: $\mathbb{Z}$) {
$todosEnMatrizActivados(camino, A) \wedge todosAdyacentes(camino, k)$
}
`pred todosEnMatrizActivados` (camino: sqPixel, A: imagen) {
$(\forall i : \mathbb{Z})(0 \leq i < |camino| \longrightarrow_L activado(A, camino[i]))$
}
`pred todosAdyacentes` (camino: $seq\langle pixel \rangle$, k: $\mathbb{Z}$) {
$(\forall i : \mathbb{Z})(1 \leq i < |camino| \longrightarrow_L esAdyacente(camino[i-1], camino[i], k))$
}
`pred esAdyacente` (p: pixel, q: pixel, k: $\mathbb{Z}$) {
$(k = 4 \wedge esAdyacente4(p, q)) \vee (k = 8 \wedge esAdyacente8(p, q))$
}
`pred esAdyacente4` (p: pixel, q: pixel) {
$abs(p[0] - q[0]) + abs(p[1] - q[1]) \leq 1$
}
`pred esAdyacente8` (p: pixel, q: pixel) {
$max(abs(p[0] - q[0]), abs(p[1] - q[1])) \leq 1$

}
aux abs (a: $\mathbb{Z}$) : $\mathbb{Z}$ = if $a > 0$ then a else $-a$ fi;
aux max (a: $\mathbb{Z}$, b: $\mathbb{Z}$) : $\mathbb{Z}$ = if $a > b$ then a else b fi;
aux min (a: $\mathbb{Z}$, b: $\mathbb{Z}$) : $\mathbb{Z}$ = if $a > b$ then b else a fi;

4.2.3. Usados desde ejercicio 3

pred hayDato (A: imagen) {
$(\exists\ p : pixel)(pixelValido(p, A) \wedge_L activado(p, A))$
}
pred imagenApagada (A:imagen) {
$\neg hayDato(A)$
}
pred esListaRegiones (lR: $seq\langle sqPixel\rangle$, A: imagen, k: $\mathbb{Z}$) {
$estanTodasLasRegiones(lR, A) \wedge_L sonRegionesValidas(lR, A, k) \wedge_L todasRegionesDiferentes(lR, A, k))$
}
pred estanTodasLasRegiones (lR: $seq\langle sqPixel\rangle$, A: imagen) {
$(\exists\ sA : sqPixel)\ (cumpleDualidadImagenSecuencia(sA, A) \wedge_L$
$(\forall\ p : pixel)p \in sA \leftrightarrow ((\exists\ sR : sqPixel)sR \in lR \wedge p \in sR)$
}
pred sonRegionesValidas (lR: $seq\langle sqPixel\rangle$, A: image, k: $\mathbb{Z}$) {
$(\forall\ region : sqPixel)(region \in lR \longrightarrow esRegion(region, A, k))$
}
pred esRegion (sR: sqPixel, A: image, k: $\mathbb{Z}$) {
$|region| > 0 \wedge ((\forall\ q : pixel)(\forall\ p : pixel)(p \in sR \wedge q \in sR) \longrightarrow estanConectados(A, p, q, k))$
}
pred todasRegionesDiferentes (lR: $seq\langle sqPixel\rangle$, k: $\mathbb{Z}$) {
$(\forall\ i : \mathbb{Z})(\forall j : \mathbb{Z})\ (0 \leq i < |lR| \wedge 0 \leq j < |lR| \wedge i \neq j \longrightarrow_L \neg sonAdyacentes(lR[i], lR[j], k))$
}
pred sonAdyacentes (region1: sqPixel, region2: sqPixel, k: $\mathbb{Z}$) {
$(\exists\ p : pixel)(\exists\ q : pixel)(p \in region1 \wedge q \in region2) \wedge_L esAdyacente(p, q, k)))$
}

4.2.4. Usados desde ejercicio 5

pred esElementoEstructuranteValido (B: imagen) {
$esImgValida(B) \wedge_L |B| = |B[0]| \wedge |B|\ \%2 = 1$
}
pred mismasDimensiones (I: imagen, J:imagen) {
$|I| = |J| \wedge_L |I[0]| = |J[0]|$
}
pred esSeqBDesplazada (B: imagen, sqB: sqPixel, z: pixel) {
$(\exists\ sBtmp : sqPixel)\ (cumpleDualidadImagenSecuencia(sBtmp, B) \wedge_L esVersionDesplazadaEnZ(sqB, sBtmp, B, z))$
}
pred esVersionDesplazadaEnZ (sqDesplazada: sqPixel, sqOriginal: sqPixel, B: imagen, z: pixel) {
$|sqDesplazada| = |sqOriginal| \wedge$
$(\forall\ p : pixel)(p \in sqOriginal \leftrightarrow \langle p[0] + z[0] - radioB(|B|),\ p[1] + z[1] - radioB(|B|)\rangle \in sqDesplazada)$
}
pred esDilatacion (A: imagen, B: imagen, D: imagen) {
$esImgValida(D) \wedge mismasDimensiones(A, D) \wedge (\exists\ sqA : sqPixel)\ ($
$cumpleDualidadImagenSecuencia(sqA, A) \wedge sonTodosLosCentrosDeDiscosQueIntersecan(sqA, B, D)$
)
}
pred sonTodosLosCentrosDeDiscosQueIntersecan (sqA: sqPixel, B: imagen, D: imagen) {
$(\forall\ z : pixel)\ (esCoordenada(z) \wedge_L pixelEnRango(z, D) \longrightarrow_L ($
$activado(z, D) \leftrightarrow elementoDesplazadoIntersecaConImagen(B, sqA, z)$
)
}
pred elementoDesplazadoIntersecaConImagen (B: imagen, sqIn: sqPixel, z: pixel) {
$(\exists\ sBp : sqPixel)\ (esSeqBDesplazada(B, sBp, z) \wedge hayInterseccion(sqIn, sBp))$
}
pred estaContenida (sqA: sqPixel, sqB:sqPixel) {
$(\forall p : pixel)(p \in sqA \rightarrow p \in sqB)$

}
`aux radioB` (w: $\mathbb{Z}$) : $\mathbb{Z}$ = w div 2 **;**

4.2.5. Usados desde ejercicio 6

`pred esElementoEstructuranteValidoK8` (B: imagen) {
$esElementoEstructuranteValido(B) \wedge$
$(\forall p : pixel) pixelValido(p, B) \longrightarrow_L activado(p, B)$
}