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

Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2020

Versión 1: 28 de octubre de 2020

TPI - "Juego de la Vida Toroidal"

Entrega: 13 de Noviembre (hasta las 17 hs)

1. Consignas

- Implementar todas las funciones que se encuentran en el archivo `ejercicios.h`. Para ello, deberán usar la especificación que se encuentra en la sección 3 del presente enunciado. La implementación de cada ejercicio DEBE SEGUIR OBLIGATORIAMENTE ESTA ESPECIFICACIÓN.
- Extender el conjunto de casos de tests de manera tal de lograr una cobertura de líneas mayor al 95 %. 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**, **pair**, 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 las definiciones de los tipos del TPI.
- `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 `tpi-template-alumnos`.
  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.

2. Entregable

La fecha de entrega del TPI es el **13 de Noviembre de 2020**.

1. Entregar una implementación de los ejercicios que cumplan el comportamiento detallado en la Especificación. Los archivos obligatorios que se deben entregan son: `ejercicios.cpp`, `auxiliares.cpp`, `auxiliares.h` y el `CMakeList.txt`. Además, incluir los casos de test adicionales propuestos por el grupo que debieron desarrollar para incrementar la cobertura.
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.**.
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.**

3. Especificación

Implementar funciones en C++ que cumplan las siguientes especificaciones respetando el **renombre** de tipo:

type *toroide*= $seq\langle seq\langle \mathsf{Bool}\rangle\rangle$

type *rectangulo*= $seq\langle seq\langle \mathsf{Bool}\rangle\rangle$

3.1. Ejercicios

Ejercicio 1.

`proc toroideValido` (in t: *toroide*, out result: Bool) {

`Pre` $\{True\}$

`Post` $\{result = true \leftrightarrow esToroide(t)\}$

}

Ejercicio 2.

`proc posicionesVivas` (in t: *toroide*, out vivas: $seq\langle \mathbb{Z} \times \mathbb{Z}\rangle$) {

`Pre` $\{esToroide(t)\}$

`Post` $\{sinRepetidos(vivas) \wedge posicionesValidas(vivas, t) \wedge_L estanLasVivas(vivas, t) \wedge cantidadVivas(t) = |vivas|\}$

`pred posicionesValidas` (listapos: $seq\langle \mathbb{Z} \times \mathbb{Z}\rangle$, t: *toroide*) {

$(\forall p : \mathbb{Z} \times \mathbb{Z})(p \in listapos \longrightarrow_L enRangoToroide(p_0, p_1, t))$

}

`pred estanLasVivas` (vivas: $seq\langle \mathbb{Z} \times \mathbb{Z}\rangle$, t: *toroide*) {

$(\forall p : \mathbb{Z} \times \mathbb{Z})(p \in vivas \leftrightarrow estaViva(p_0, p_1, t))$

}

}

Ejercicio 3.

`proc densidadPoblacion` (in t: *toroide*, out result : $\mathbb{R}$) {

`Pre` $\{esToroide(t)\}$

`Post` $\{result = cantidadVivas(t)/superficieTotal(t)\}$

}

Ejercicio 4.

`proc evolucionDePosicion` (in t: *toroide*, in pos : $\mathbb{Z} \times \mathbb{Z}$, out result : Bool) {

`Pre` $\{esToroide(t) \wedge enRangoToroide(pos_0, pos_1, t)\}$

`Post` $\{result = true \leftrightarrow debeVivir(t, pos_0, pos_1)\}$

}

Ejercicio 5.

`proc evolucionToroide` (inout t: *toroide*) {

`Pre` $\{esToroide(T_0) \wedge t = T_0\}$

Post $\{esEvolucionToroide(t, T_0)\}$

}

Ejercicio 6.

proc evolucionMultiple (in t: *toroide*, in k: $\mathbb{Z}$, out result: *toroide*) {

Pre $\{esToroide(t) \land k \geq 1\}$

Post $\{esEvolucionNivelK(result, t, k)\}$

}

Ejercicio 7.

proc esPeriodico (in t: *toroide*, inout p: $\mathbb{Z}$, out result: Bool) {

Pre $\{esToroide(t) \land p = P_0\}$

Post $\{(cumpleEvolucionCiclica(t) \rightarrow result = true \land tienePeriodoP(t, p)) \land$
$(\neg cumpleEvolucionCiclica(t) \rightarrow (result = false \land p = P_0))\}$

pred cumpleEvolucionCiclica (t: *toroide*) {

$(\exists k : \mathbb{Z})(k > 0 \land esEvolucionNivelK(t, t, k))$

}

pred tienePeriodoP (t: *toroide*, p: $\mathbb{Z}$) {

$esEvolucionNivelK(t, t, p) \land (\forall q : \mathbb{Z})(1 \leq q < p \rightarrow \neg esEvolucionNivelK(t, t, q))$

}

}

Ejercicio 8.

proc primosLejanos (in t1: *toroide*, in t2: *toroide*, out primos: Bool) {

Pre $\{esToroide(t1) \land esToroide(t2) \land_L mismaDimension(t1, t2)\}$

Post $\{primos = true \leftrightarrow (\exists k : \mathbb{Z})(k > 0 \land_L esEvolucionNivelK(t2, t1, k) \lor esEvolucionNivelK(t1, t2, k))\}$

}

Ejercicio 9.

proc seleccionNatural (in ts: $seq\langle toroide\rangle$, out res: $\mathbb{Z}$) {

Pre $\{|ts| > 0 \land_L todosValidos(ts) \land todosVivos(ts) \land todosSeExtinguen(ts)\}$

Post $\{0 \leq res < |ts| \land_L sobreviveATodos(res, ts)\}$

pred todosVivos (ts: $seq\langle toroide\rangle$) {

$(\forall i : \mathbb{Z})(0 \leq i < |ts| \longrightarrow_L cantidadVivas(ts[i]) > 0)$

}

pred todosSeExtinguen (ts: $seq\langle toroide\rangle$) {

$(\forall i : \mathbb{Z})(\ 0 \leq i < |ts| \longrightarrow_L (\exists k : \mathbb{Z})(k > 0 \land_L seExtingueEnK(ts[i], k))\ )$

}

pred seExtingueEnK (t: *toroide*, k: $\mathbb{Z}$) {

$(\exists tmuerto : toroide)(esToroide(tmuerto) \land mismaDimension(t, tmuerto) \land cantidadVivas(tmuerto) = 0 \land_L$
$esEvolucionNivelK(tmuerto, t, k) \land (\forall k' : \mathbb{Z})(0 \leq k' < k \longrightarrow_L \neg esEvolucionNivelK(tmuerto, t, k')))$

}

pred sobreviveATodos (i: $\mathbb{Z}$, ts: $seq\langle toroide\rangle$) {

$(\forall j : \mathbb{Z})(0 \leq j < i \longrightarrow_L \neg muereAntes(ts[i], ts[j])) \wedge$
$(\forall h : \mathbb{Z})(i < h < |ts| \longrightarrow_L muereAntes(ts[h], ts[i]))$

}

`pred muereAntes` (tin: *toroide*, tduracell: *toroide*) {

$(\exists k1 : \mathbb{Z})((\exists k2 : \mathbb{Z})(seExtingueEnK(tin, k1) \wedge seExtingueEnK(tduracell, k2) \wedge k1 \leq k2))$

}

}

Ejercicio 10.

`proc fusionar` (in t1: *toroide*, in t2: *toroide*, out res: *toroide*) {

`Pre` $\{esToroide(t1) \wedge esToroide(t2) \wedge mismaDimension(t1, t2)\}$

`Post` $\{esToroide(res) \wedge mismaDimension(res, t1) \wedge_L interseccionVivas(res, t1, t2)\}$

`pred interseccionVivas` (tf: *toroide*, t1: *toroide*, t2: *toroide*) {

$(\forall f : \mathbb{Z})(\forall c : \mathbb{Z})(enRangoToroide(f, c, tf) \rightarrow_L (estaViva(f, c, tf) \leftrightarrow estaViva(f, c, t1) \wedge estaViva(f, c, t2))))$

}

}

Ejercicio 11.

`proc vistaTrasladada` (in t1: *toroide*, in t2: *toroide*, out res: Bool) {

`Pre` $\{esToroide(t1) \wedge esToroide(t2) \wedge_L mismaDimension(t1, t2)\}$

`Post` $\{res = true \leftrightarrow esTrasladada(t1, t2)\}$

}

Ejercicio 12.

`proc menorSuperficieViva` (in t: *toroide*, out res: $\mathbb{Z}$) {

`Pre` $\{esToroide(t) \wedge_L cantidadVivas(t) > 0\}$

`Post` $\{(\exists sr0 : rectangulo)(\exists t0 : toroide)(esSubRecToroideValido(sr0, t0, t) \wedge_L$
$superficieTotal(sr0) = res \wedge (\forall sr1 : rectangulo)(\forall t1 : toroide)(esSubRecToroideValido(sr1, t1, t) \longrightarrow_L$
$superficieTotal(sr1) \geq res))\}$

`pred esSubRecToroideValido` (subrec: *rectangulo*, tt: *toroide*, torig: *toroide*) {

$dimensionesValidasSubRec(subrec, tt) \wedge esTrasladada(tt, torig) \wedge_L cubreLosVivos(subrec, tt)$

}

`pred dimensionesValidasSubRec` (subrec: *rectangulo*, t: *toroide*) {

$esRectangulo(subrec) \wedge_L filas(subrec) \leq filas(t) \wedge columnas(subrec) \leq columnas(t)$

}

`pred cubreLosVivos` (subrec: *rectangulo*, t: *toroide*) {

$(\exists i : \mathbb{Z})(\exists j : \mathbb{Z})(desplazamientoValido(i, j, subrec, t) \wedge_L cantidadVivas(t) = cantidadVivas(subrec))$

}

`pred desplazamientoValido` (i: $\mathbb{Z}$, j: $\mathbb{Z}$, subrec: *rectangulo*, t: *toroide*) {

$(\forall f : \mathbb{Z})(\forall c : \mathbb{Z})(enRangoToroide(f, c, subrec) \wedge enRangoToroide(f + i, c + j, t)$
$\longrightarrow_L estaViva(f, c, subrec) \leftrightarrow estaViva(f + i, c + j, t))$

}

}

3.2. Predicados y funciones auxiliares

aux cantidadVivas (t: $seq\langle seq\langle \text{Bool}\rangle\rangle$) : $\mathbb{Z} = \sum_{f=0}^{filas(t)-1} \sum_{c=0}^{columnas(t)-1}$ if $estaViva(f,c,t)$ then 1 else 0 fi ;

aux columnas (t: $seq\langle seq\langle \text{Bool}\rangle\rangle$) : $\mathbb{Z}$ = if $filas(t) > 0$ then $|t[0]|$ else 0 fi ;

aux columnaToroide (c: $\mathbb{Z}$, t: *toroide*) : $\mathbb{Z} = c \ mod \ columnas(t)$;

pred debeVivir (t: *toroide*, f: $\mathbb{Z}$, c: $\mathbb{Z}$) {

$(estaViva(f,c,t) \rightarrow 2 \leq vecinosVivos(t,f,c) \leq 3) \wedge (estaMuerta(f,c,t) \rightarrow vecinosVivos(t,f,c) = 3)$

}

pred enRango (i: $\mathbb{Z}$, s: $seq\langle T\rangle$) {

$0 \leq i < |s|$

}

pred enRangoToroide (f: $\mathbb{Z}$, c: $\mathbb{Z}$, t: $seq\langle seq\langle \text{Bool}\rangle\rangle$) {

$enRango(f,t) \wedge_L enRango(c,t[f])$

}

pred esEvolucionNivelK (tf: *toroide*, ti: *toroide*, k: $\mathbb{Z}$) {

$k \geq 0 \wedge_L (\exists s : seq\langle toroide\rangle)(todosValidos(s) \wedge |s| = k+1 \wedge_L s[0] = ti \wedge sonTicksConsecutivos(s) \wedge s[k] = tf)$

}

pred esEvolucionToroide (tf: *toroide*, ti: *toroide*) {

$mismaDimension(tf,ti) \wedge_L (\forall f : \mathbb{Z})((\forall c : \mathbb{Z})(enRangoToroide(f,c,ti) \rightarrow_L (estaViva(f,c,tf) \leftrightarrow debeVivir(ti,f,c))))$

}

pred esRectangulo (r: $seq\langle seq\langle \text{Bool}\rangle\rangle$) {

$filas(r) > 0 \wedge columnas(r) > 0 \wedge (\forall f : \mathbb{Z})(0 \leq f < |r| \rightarrow_L |r[f]| = |r[0]|)$

}

pred estaMuerta (f: $\mathbb{Z}$, c: $\mathbb{Z}$, t: *toroide*) {

$enRangoToroide(f,c,t) \wedge_L t[f][c] = false$

}

pred estaViva (f: $\mathbb{Z}$, c: $\mathbb{Z}$, t: $seq\langle seq\langle \text{Bool}\rangle\rangle$) {

$enRangoToroide(f,c,t) \wedge_L t[f][c] = true$

}

pred esToroide (t: *toroide*) {

$esRectangulo(t) \wedge filas(t) \geq 3 \wedge columnas(t) \geq 3$

}

pred esTrasladada (t1: *toroide*, t2: *toroide*) {

$(\exists i : \mathbb{Z})(\exists j : \mathbb{Z})(traslacion(t1,t2,i,j))$

}

aux filas (t: $seq\langle seq\langle \text{Bool}\rangle\rangle$) : $\mathbb{Z} = |t|$;

aux filaToroide (f: $\mathbb{Z}$, t: *toroide*) : $\mathbb{Z} = f \ mod \ filas(t)$;

pred menorSuperficie (st: *toroide*, t: *toroide*) {

$superficieTotal(st) \leq superficieTotal(t)$

}

pred mismaCantidadVivas (st: *toroide*, t: *toroide*) {

$cantidadVivas(st) = cantidadVivas(t)$

}

pred mismaDimension (t1: *toroide*, t2: *toroide*) {

$filas(t1) = filas(t2) \land columnas(t1) = columnas(t2)$

}

pred posValida (t: *toroide*, p: $\mathbb{Z} \times \mathbb{Z}$) {

$enRangoToroide(p_0, p_1, t)$

}

pred sinRepetidos (s: $seq\langle T \rangle$) {

$(\forall i : \mathbb{Z})(enRango(i, s) \rightarrow_L \#apariciones(s, s[i]) = 1)$

}

pred sonTicksConsecutivos (ts: $seq\langle toroide \rangle$) {

$(\forall i : \mathbb{Z})(0 \leq i < |ts| - 1 \rightarrow_L esEvolucionToroide(ts[i+1], s[i]))$

}

pred todosValidos (ts: $seq\langle toroide \rangle$) {

$(\forall i : \mathbb{Z})(0 \leq i < |ts| \longrightarrow_L esToroide(ts[i]))$

}

aux superficieTotal (t: *toroide*) : $\mathbb{Z} = filas(t) * columnas(t)$;

pred traslacion (t1: *toroide*, t2: *toroide*, i: $\mathbb{Z}$, j: $\mathbb{Z}$) {

$(\forall f : \mathbb{Z})(\forall c : \mathbb{Z})(enRangoToroide(f, c, t1) \longrightarrow_L estaViva(f, c, t1) \leftrightarrow vivaToroide(f + i, c + j, t2))$

}

pred vecinaViva (t: *toroide*, f: $\mathbb{Z}$, c: $\mathbb{Z}$, i: $\mathbb{Z}$, j: $\mathbb{Z}$) {

$vivaToroide(f + i, c + j, t)$

}

aux vecinosVivos (t: *toroide*, f: $\mathbb{Z}$, c: $\mathbb{Z}$) : $\mathbb{Z} = \sum_{i=-1}^{1} \sum_{j=-1}^{1}$ if $(i \neq 0 \lor j \neq 0) \land vecinaViva(t, f, c, i, j)$ then 1 else 0 fi;

pred vivaToroide (f: $\mathbb{Z}$, c: $\mathbb{Z}$, t: *toroide*) {

$estaViva(filaToroide(f, t), columnaToroide(c, t), t)$

}

4. Funciones C++

La declaración de las funciones a implementar es la siguiente:

```
bool toroideValido(vector<vector<bool>> const &t);
vector<posicion> posicionesVivas(toroide const &t);
float densidadPoblacion(toroide const &t);
bool evolucionDePosicion(toroide const &t, posicion x);
void evolucionToroide(toroide &t);
toroide evolucionMultiple(toroide const &t, int K);
bool esPeriodico(toroide const &t, int &p);
bool primosLejanos(toroide const &t, toroide const &u);
int seleccionNatural(vector <toroide> ts);
toroide fusionar(toroide const &t, toroide const &u);
bool vistaTrasladada(toroide const &t, toroide const &u);
int menorSuperficieViva(toroide const &t);
```

Donde definimos las siguientes estructuras de datos

```
typedef vector<vector <bool>> toroide;
typedef pair<int, int> posicion;
```

Uso de pair Vamos a utilizar el container pair que pertenece a la librería estándar del C++ y está definido en el header **utility**. Este es un container que puede poner juntos dos elementos de cualquier tipo. En nuestro caso, vamos a utilizarlo para dos valores enteros.

Para asignar u acceder al primero se utiliza la propiedad *first*, y para el otro... *second*. En el siguiente ejemplo[1], veremos varias maneras de declarar, asignar e imprimir los valores de containers pair.

```
#include <iostream>
#include <utility>
using namespace std;

int main()
{
    pair <int, char> PAIR1 ;
    pair <string, double> PAIR2 ("GeeksForGeeks", 1.23) ;
    pair <string, double> PAIR3 ;

    PAIR1.first = 100;
    PAIR1.second = 'G' ;

    PAIR3 = make_pair ("GeeksForGeeks␣is␣Best",4.56);

    cout << PAIR1.first << "␣" ; // espacio en blanco para separar los elementos
    cout << PAIR1.second << endl ;

    cout << PAIR2.first << "␣" ;
    cout << PAIR2.second << endl ;

    cout << PAIR3.first << "␣" ;
    cout << PAIR3.second << endl ;

    return 0;
}
```

[1] https://www.geeksforgeeks.org/pair-in-cpp-stl/