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

Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2022

Versión 2: 21 de Octubre de 2022

Trabajo Práctico de Implementación

Buscaminas

Entrega: 4 de Noviembre de 2022 (hasta las 23:55)

1. Introducción

En el TPE hemos desarrollado una especificación para el juego del buscaminas. ¡Llegó la hora de implementarlo! En este TP realizaremos una implementación en C++ de los procedimientos especificados en el TPE.

En este caso, no van a implementar su propia especificación, sino que tendrán que basarse en la Especificación de la Cátedra, que se encuentra más abajo. Antes de empezar a programar léanla con atención y marquen las diferencias con la de ustedes, si es que existen, para evitar errores de interpretación.

Si bien no vamos a pedirles algorítmos óptimos, van a tener que calcular la complejidad temporal de las funciones que implementen y proponer alguna mejora.

Finalmente, como bien sabrán a esta altura, una implementación no está completa si no cuenta con una adecuado conjunto de tests. En este TP van a tener que escribir tests para cada una de las funciones. Además de escribir tests que cubran casos borde y datos que consideren que pueden traer problemas, les vamos a pedir un cierto cubrimiento mínimo de líneas.

2. Consignas

- Implementar todas las funciones que se encuentran en el archivo `ejercicios.h`. Para ello, deberán usar la especificación que se encuentra más abajo. La implementación deberá ser prolija: deberán evitar repetir implementaciones innecesariamente y deberán usar adecuadamente funciones auxiliares.

  Las funciones a implementar son las siguientes:

```
int minasAdyacentes(tablero& t, pos p);
void cambiarBanderita(tablero& t, jugadas& j, pos p, banderitas& b);
bool perdio(tablero& t, jugadas& j);
bool gano(tablero& t, jugadas& j);
void jugarPlus(tablero& t, banderitas& b, pos p, jugadas& j);
bool sugerirAutomatico121(tablero& t, banderitas& b, jugadas& j, pos& p);
```

- Escribir casos de tests para todas las funciones desarrolladas de manera tal de lograr una cobertura de líneas mayor al 95 %. En caso de no poder alcanzarla, explicar el motivo. Para calcular la cobertura pueden usar CLion (como vimos en las clases de labo)
- Calcular la complejidad temporal de peor caso de todos los procedimientos de `ejercicios.h`. Anótenla como comentarios al inicio de cada función y también en las partes relevantes de los algoritmos, junto con una breve justificación.
- Responder la siguiente pregunta: ¿qué pasaría si la estructura `jugadas` fuera una matriz conteniendo en cada posición la cantidad de minas adyacentes (o -1 si la casilla no fue jugada)? ¿Cómo afectaría eso a la complejidad temporal de peor caso de los procedimientos `jugarPlus` y `sugerirAutomatico121`? Escriban la respuesta en el archivo `RESPUESTAS.md`.

3. Observaciones

- La evaluación tendrá en consideración los siguientes aspectos:
  - Que compile y que pase todos los tests implementados por ustedes
  - Que los tests tengan el cubrimiento requerido
  - Que el programa pase los tests de la cátedra (a los cuáles ustedes no van a tener acceso)
  - Que el código no tenga errores y la implementación sea prolija
- No está permitido el uso de librerías de C++ fuera de las ya utilizadas en los laboratorios. Consultar con la cátedra por cualquier librería adicional.
- Por un tema de eficiencia, las estructuras grandes se pasan por referencia. Tengan cuidado de no modificar los datos si éstas son parámetros de entrada
- Tengan en cuenta que la descomposición en funciones auxiliares no tiene por qué ser la misma que la que hicieron en la especificación. En la especificación descomponemos únicamente para que sea más fácil leer y comprender los predicados, aquí tenemos en cuenta también la reutilización de código y la eficiencia de los algoritmos.

4. Entregable

La fecha de entrega del TPI es el **4 de NOVIEMBRE de 2022**.

El entregable debe estar compuesto por un archivo comprimido conteniendo una carpeta con todo el código fuente, tanto el de ustedes como el del template, incluyendo `CMakeLists.txt` y `RESPUESTAS.md`. No deberán incluir código objeto, archivos temporales ni ejecutables (como por ejemplo la carpeta `cmake-build-debug`).

5. Especificación

5.1. Tipos

type $pos = \mathbb{Z} \times \mathbb{Z}$

type $tablero = seq\langle seq\langle \mathsf{Bool} \rangle \rangle$

type $jugada = pos \times \mathbb{Z}$

type $jugadas = seq\langle pos \times \mathbb{Z} \rangle$

type $banderitas = seq\langle pos \rangle$

type $posiciones = seq\langle pos \rangle$

5.2. Problemas

Ejercicio 1. proc minasAdyacentes(in t: *tablero*, in p: *pos*, out res: $\mathbb{Z}$)
Dado un tablero válido t y una posición válida p, esta función devuelve en res la cantidad de minas adyacentes a p, de acuerdo con las reglas del juego. Dos casilleros se consideran adyacentes si se tocan de forma vertical, horizontal o diagonal.

```
proc minasAdyacentes (in t: tablero, in p: pos, out res: Z) {
    Pre {tableroValido(t) ∧ posicionValida(p, |t|)}
    Post {res = numMinasAdyacentes(t, p)}
}
pred tableroValido (t: tablero) {
    |t| > 0 ∧_L (∀j : Z)(0 ≤ i ≤ |t| →_L |t[i]| = |t|) ∧ cantidadMinasValida(t)
}
pred cantidadMinasValida (t: tablero) {
    0 < cantidadMinas(t) < |t|^2
}
```

aux cantidadMinas (t: *tablero*) : $\mathbb{Z} = \sum_{f=0}^{|t|} \sum_{c=0}^{|t[0]|}$ if $t[f][c] = true$ then 1 else 0 fi ;

pred posicionValida (p: *pos*, n: $\mathbb{Z}$) {

$coordenadaValida(p_0, n) \wedge coordenadaValida(p_1, n)$

}

pred coordenadaValida (c: $\mathbb{Z}$, n: $\mathbb{Z}$) {

$0 \leq c < n$

}

aux numMinasAdyacentes (t: *tablero*, p: *pos*) : $\mathbb{Z}$ =
$(\sum_{i=-1}^{1} \sum_{j=-1}^{1}$ if $esAdyacenteValida(p, i, j, t) \wedge_L t[p_0 + i][p_1 + j] = true$ then 1 else 0 fi) ;

pred esAdyacenteValida (p, i, j, t) {

$coordenadaValida(p_0 + i, |t|) \wedge coordenadaValida(p_1 + j, |t[0]|) \wedge (i \neq 0 \vee j \neq 0)$

}

Ejercicio 2. proc cambiarBanderita(in t: *tablero*, in j: *jugadas*, in p: *pos*, inout bs: *banderitas*)
El jugador marca la posición p con una banderita en el ayuda-memoria. No se puede plantar una banderita en un casillero ya descubierto ni en un casillero que ya tiene una banderita. Si ya hay banderita en la posición p, la misma es eliminada.

proc cambiarBanderita (in t: *tablero*, in j: *jugadas*, in p: *pos*, inout bs: *banderitas*) {

Pre $\{juegoValido(t, j) \wedge_L banderitasValidas(bs, t, j) \wedge (esPosicionSinJugarYSinBanderita(bs, j, p)$
$\vee esBanderita(p, bs)) \wedge bs = bs_0\}$

Post $\{(esBanderita(p, bs_0) \rightarrow sacaBanderita(p, bs, bs_0)) \wedge$
$(\neg esBanderita(p, b_0) \rightarrow plantaBanderita(p, bs, bs_0))\}$

}

pred juegoValido (t: *tablero*, j: *jugadas*) {

$tableroValido(t) \wedge_L jugadasValidas(j, t) \wedge minasPisadas(j, t) \leq 1$
$\wedge jugadasNoRepetidas(j)$

}

pred jugadasValidas (j: *jugadas*, t: *tablero*) {

$(\forall i : \mathbb{Z})(0 \leq i < |j| \rightarrow_L (posicionValida(j[i]_0, |t|) \wedge j[i]_1 = numMinasAdyacentes(j[i]_0, t)))$

}

aux minasPisadas (j: *jugadas*, t: *tablero*) : $\mathbb{Z} = \sum_{i=0}^{|j|}$ if $hayMinaEnPosicion(j[i]_0, t)$ then 1 else 0 fi ;

pred hayMinaEnPosicion (p: *pos*, t: *tablero*) {

$t[p_0][p_1] = true$

}

pred jugadasNoRepetidas (j: *jugadas*) {

$(\forall i : \mathbb{Z})(\forall k : \mathbb{Z})(0 \leq i, k < |j| \wedge i \neq k \rightarrow_L j[i]_0 \neq j[k]_0)$

}

pred banderitasValidas (b: *banderitas*, t: *tablero*, j: *jugadas*) {

$(\forall p : pos)(esBanderita(p, b) \rightarrow (posicionValida(p, |t|) \wedge \neg fueJugada(p, j))) \wedge banderitasSinRepetir(b)$

}

pred esBanderita (p: pos, b: *banderitas*) {

$(\exists i : \mathbb{Z})(0 \leq i < |b| \wedge_L p = b[i])$

}

pred banderitasSinRepetir (b: *banderitas*) {

$(\forall p : pos)(esBanderita(p, b) \rightarrow_L \#apariciones(p, b) = 1)$

}

pred fueJugada (p: pos, j: *jugadas*) {

$(\exists i : \mathbb{Z})(0 \leq i < |j| \wedge_L p = j[i]_0)$

}

pred esPosicionSinJugarYSinBanderita (p: *pos*, j: *jugadas*, b: *banderitas*, t: *tablero*) {

$posicionValida(p, |t|) \wedge \neg fueJugada(p, j) \wedge \neg esBanderita(p, b)$

}

pred sacaBanderita (posSacar: *pos*, bs: *banderitas*, bs_0: *banderitas*) {

$|bs| = |bs_0| - 1 \wedge (\forall b : pos)((esBanderita(b, bs_0) \wedge \neg(posSacar = b)) \rightarrow esBanderita(b, bs))$

}

pred plantaBanderita (posPlantar: *pos*, bs: *banderitas*, bs_0: *banderitas*) {

$|bs| = |bs_0| + 1 \wedge (\forall b : pos)(esBanderita(b, bs_0) \rightarrow esBanderita(b, bs)) \wedge esBanderita(posPlantar, bs)$

}

Ejercicio 3. proc perdió(in t: *tablero*, in j: *jugadas*, out res: Bool)
Devuelve res = true si el jugador ha perdido el juego y res = false si el jugador no ha perdido.

proc perdio (in t: *tablero*, in j: *jugadas*, out res: Bool) {

Pre $\{juegoValido(j, t)\}$

Post $\{res = true \leftrightarrow juegoPerdido(t, j)\}$

}

pred juegoPerdido (t: *tablero*, js: *jugadas*) {

$(\exists j : jugada)(j \in js \wedge_L hayMinaEnPosicion(j_0, t))$

}

Ejercicio 4. proc ganó(in t: *tablero*, in j: *jugadas*, out res: Bool)
Devuelve res = true si el jugador ha ganado el juego y res = false si el jugador no ha ganado.

proc gano (in t: *tablero*, in j: *jugadas*, out res: Bool) {

Pre $\{juegoValido(j, t)\}$

Post $\{res = true \leftrightarrow juegoGanado(t, j)\}$

}

pred juegoGanado (t: *tablero*, j: *jugadas*) {

$(\forall p : pos)(posicionValida(p, |t|) \rightarrow_L (t[p_0][p_1] = false \leftrightarrow fueJugada(p, j)))$

}

Ejercicio 5. proc jugarPlus(in t: *tablero*, in b: *banderitas*, in p: *pos*, inout j: *jugadas*)

proc jugarPlus (in t: *tablero*, in b: *banderitas*, in p: *pos*, inout j: *jugadas*) {

Pre $\{juegoValido(t, j) \wedge_L banderitasValidas(b, t, j) \wedge esPosicionSinJugarYSinBanderita(p, j, b, t) \wedge \neg juegoGanado(t, j) \wedge \neg juegoPerdido(t, j) \wedge j = j_0\}$

Post $\{mantieneJugadas(j_0, j) \wedge jugadasNoRepetidas(j) \wedge incluyeJugadaActual(t, j, p) \wedge soloAgregaPosicionesDescubiertas(p, j, j_0, b, t)\}$

}

pred mantieneJugadas (js_0: *jugadas*, js: *jugadas*) {

$(\forall j : jugada)(j \in js_0 \rightarrow_L j \in js)$

}

`pred incluyeJugadaActual` (t: *tablero*, j: *jugadas*, p: *pos*) {

$(p, minasAdyacentes(t, p)) \in j$

}

`pred soloAgregaPosicionesDescubiertas` (p: *pos*, j: *jugadas*, j_0: *jugadas*, b: *banderitas*, t: *tablero*) {

$descubreSoloPosicionJugada(p, j, j_0, t) \vee descubreMultiplesPosiciones(p, j, j_0, b, t)$

}

`pred descubreSoloPosicionJugada` (p: *pos*, j: *jugadas*, j_0: *jugadas*, t: *tablero*) {

$(hayMinaEnPosicion(p, t) \vee numMinasAdyacentes(p, t) > 0) \wedge |j| = |j_0| + 1$

}

`pred descubreMultiplesPosiciones` (posJugada: *pos*, j: *jugadas*, j_0: *jugadas*, b: *banderitas*, t: *tablero*) {

$(\neg(hayMinaEnPosicion(posJugada, t)) \wedge numMinasAdyacentes(posJugada, t) = 0) \wedge$
$(incluyeDescubiertasAutomaticamente(posJugada, j, j_0, b, t)$
$\wedge |j| = |j_0| + 1 + \#descubiertasAutoDesde(posJugada, j_0, j, b, t))$

}

`pred incluyeDescubiertasAutomaticamente` (p: *pos*, j: *jugadas*, j_0: *jugadas*, b: *banderitas*, t: *tablero*) {

$(\forall q : pos)(esDescubiertaAuto(q, p, j, j_0, b, t) \rightarrow_L incluyeJugadaActual(t, j, q))$

}

`pred esDescubiertaAuto` (posDescubierta: *pos*, posJugada: *pos*, j: *jugadas*, j_0: *jugadas*, b: *banderitas*, t: *tablero*) {

$posicionValida(posDescubierta, |t|) \wedge_L hayCaminoLibre(posJugada, posDescubierta, j_0, b, t)$

}

`pred hayCaminoLibre` (posInicial: *pos*, posFinal: *pos*, j_0: *jugadas*, b: *banderitas*, t: *tablero*) {

$(\exists s : posiciones)(|s| > 1 \wedge_L s[0] = posInicial \wedge s[|s| - 1] = posFinal \wedge \neg esBanderita(posFinal, b) \wedge$
$\neg fueJugada(posFinal, j) \wedge \neg hayMinaEnPosicion(posFinal, t) \wedge esCaminoLibre(s, j_0, b, t))$

}

`pred esCaminoLibre` (s: *posiciones*, j_0: *jugadas*, b: *banderitas*, t: *tablero*) {

$(\forall i : \mathbb{Z})(0 \leq i < |s| - 1 \rightarrow_L (esPosicionSinJugarYSinBanderita(t, j, b, s[i]) \wedge numMinasAdyacentes(s[i], t) = 0)$
$\wedge sonPosicionesAdyacentes(s[i], s[i + 1]))$

}

`pred sonPosicionesAdyacentes` (p: *pos*, q: *pos*) {

$|p_0 - q_0| \leq 1 \wedge |p_1 - q_1| \leq 1 \wedge p \neq q$

}

`aux #descubiertasAutoDesde` (p: *pos*, j: *jugadas*, j_0: *jugadas*, b: *banderitas*, t: *tablero*) : $\mathbb{Z}$ =
$(\sum_{i=0}^{|t|-1} \sum_{j=0}^{|t|-1}$ if $esDescubiertaAuto((i, j), p, j, j_0, b, t)$ then 1 else 0 fi) ;

Ejercicio 6. proc sugerirAutomático121(in t: *tablero*, in b: *banderitas*, in j: *jugadas*, out hay: *bool*, out p: *pos*)

`proc sugerirAutomatico121` (in t: *tablero*, in b: *banderitas*, in j: *jugadas*, out hay: Bool, out p: *pos*) {

`Pre` $\{juegoValido(t, j) \wedge banderitasValidas(b, t, j)\}$

`Post` $\{(hay = true \leftrightarrow hayPosicionSugerible(j, b, t)) \wedge$
$hayPosicionSugerible(j, b, t) \rightarrow_L esPosicionSinJugarYSinBanderita(p, j, b, t) \wedge esAdyacenteA121(p, j)\}$

}

`pred hayPosicionSugerible` (j: *jugadas*, b: *banderitas*, t: *tablero*) {

$(\exists p : pos)(esPosicionSinJugarYSinBanderita(p, j, b, t)) \wedge esAdyacenteA121(p, j)$

}

`pred esAdyacenteA121` (p: *pos*, j: *jugadas*) {

$es121Horizontal((p_0 - 1, p_1), j) \vee es121Horizontal((p_0 + 1, p_1), j)$
$\vee es121Vertical((p_0, p_1 - 1), j) \vee es121Vertical((p_0, p_1 + 1), j)$

}

`pred es121Horizontal` (p: *pos*, j: *jugadas*) {

$((p_0, p_1 - 1), 1) \in j \wedge ((p_0, p_1), 2) \in j \wedge ((p_0, p_1 + 1), 1) \in j$

}

`pred es121Vertical` (p: *pos*, j: *jugadas*) {

$((p_0 - 1, p_1), 1) \in j \wedge ((p_0, p_1), 2) \in j \wedge ((p_0 + 1, p_1), 1) \in j$

}