DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP de Especificación

Buscaminas

30 de Marzo de 2022 — Algoritmos y Estructuras de Datos I

Grupo "Hello World"

Integrante	LU	Correo electrónico
Verón, Luca	208/22	luquitaver@gmail.com
Bigioli, Felipe	084/22	felipebigiolli@gmail.com
Larrea, Juan Iñaki	423/22	juaninaki@gmail.com
Wingeyer, Santiago	362/22	santiagowingeyer@yahoo.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. Introduccion

Separamos el TP en tres secciones para presentarlo de una manera mas prolija:

- Definición de Tipos
- Problemas
- Funciones y predicados de cada problema

En la sección Definición de Tipos se encuentran las definiciones de cada tipo propuestas en el enunciado del TP.
En la parte de Problemas se encuentra la resolucion los problemas.
Por último, en la sección Funciones y predicados de cada problema se encuentran las funciones y predicados que definimos para resolver los problemas, ordenados segun a que problema corresponden.

2. Definición de Tipos

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

type $tablero$= $seq\langle seq\langle Bool\rangle\rangle$

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

type $banderitas$= $seq\langle pos\rangle$

3. Problemas

3.1. Ejercicio 1

Dado un tablero valido t y una posicion valida p, esta funcion devuelve 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.

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

La funcion calcula la cantidad total de minas en los casilleros adyacentes en un rango de 3*3, con centro en la posicion elegida.

3.2. Ejercicio 2

Indica si el estado del juego representado por j es valido y se corresponde con la ubicacion real de las minas representada por un tablero t.

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

$minasEnJugadas(j, t) \leq 1 \wedge$
$jugadasValidas(t, j)$

}

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

`pred esMina` (p:*pos*, t:*tablero*) {

$t[p_0][p1] = True$

}

juegoValido recorre todas las jugadas hechas hasta el momento, y se fija que haya como maximo una mina pisada en nuestras jugadas, ya que puede pasar que el jugador no haya pisado una mina (y el juego continua) o haya pisado una mina (y el juego acaba, no puede pisar otra más).
la 2da linea verifica que la suma de minasAdyacentes se corresponda con el segundo elemento de cada jugada, y que las jugadas esten en el rango del tablero.

3.3. Ejercicio 3

El jugador marca la posicion p con una banderita en el ayuda-memoria, indicando que en su opinion en esa posicion hay una mina.

`proc plantarBanderita` (In t:*tablero*, In j:*jugadas*, In p: *pos*, Inout b: *banderitas*) {

`Pre` $\{(b = b_0 \wedge (tableroValido(t)) \wedge (juegoValido(j, t)) \wedge casilleroSinBandera(b, p)$
$\wedge casilleroNoDescubierto(j, p) \wedge_L \neg minaPisada(t, j))\}$

`Post` $\{|b| = |b_0| + 1 \wedge p \in b \wedge (\forall n : pos)(n \in b_0 \longrightarrow n \in b)\}$

}

Como precondicion definimos el inout, chequeamos que el tablero sea un tablero valido y el juego sea un juego valido. También, que no haya una bandera plantada ni que la posición a agregar la banderita ya este jugada. Por último, pedimos que no haya ninguna mina pisada, para que el jugador pueda agregar banderitas (ya que en juego válido permitimos que pueda haber una mina pisada). De haber una mina, el juego termina, el jugador perdió y no puede agregar banderitas.
Como postcondicion, pedimos que conserve los elementos anteriores y se le sume la nueva posicion.

3.4. Ejercicio 4

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 $\{tableroValido(t) \wedge_L juegoValido(j,t)\}$

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

}

En la precondicion nos fijamos que el tablero sea valido y el juego sea valido. Como post, nos fijamos si hay alguna mina pisada para devolver true o false.En caso de haber pisado una mina devuelve true y pierde el jueg.

3.5. Ejercicio 5

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 $\{tableroValido(t) \wedge_L juegoValido(j,t)\}$

Post $\{res = \text{if } todasDescubiertas(j,t) \wedge \neg minaPisada(j,t) \text{ then } true \text{ else } false \text{ fi}\}$

}

Como precondicion, nos fijamos que el tablero sea valido y que el estado del juego tambien lo sea con juegoValido. Como post, chequeamos que esten todas las casillas sin mina descubiertas (todasDescubiertas), es decir, que esten en jugadas, y negamos minaPisada, ya que para ganar no puede haber una mina pisada.

3.6. Ejercicio 6

A partir de un tablero valido y de un estado de juego valido, en el que el jugador aun no gano ni perdio, este procedimiento modifica el estado del juego tras descubrir la posicion p.

proc jugar ((In t:*tablero*, In b:*banderitas*, In p:*pos*, Inout j:*jugadas*) {

Pre $\{(j = j_0 \wedge tableroValido(t) \wedge posValida(p,t,j_0)) \wedge_L (juegoValido(j_0,t)$
$\wedge casilleroNoDescubierto(j_0,p) \wedge casilleroSinBandera(b,p) \wedge \neg minaPisada(j_0,t))\}$

Post $\{(|j| = |j_0| + 1) \wedge (p \in j) \wedge (\forall i : \mathbb{Z})(0 \leq i < |j| - 1) \longrightarrow_L (j_0[i] \in j)\}$

}

Como precondicion pedimos tableroValido y que la posicion sea valida. Luego que el juego sea un juegoValido, y que el casillero en el que queremos hacer la jugada no este descubierto, de esta forma nos aseguramos que el jugador tampoco ganó. Pedimos tambien sin bandera y que no haya minas pisadas (ya que podría haber alguna, y sería absurdo que el jugador juegue con una mina ya pisada). De postcondicion nos aseguramos que los elementos de jugadas se conserven como al inicio, y que la nueva posicion se agregue a la secuencia de posiciones jugadas.

3.7. Ejercicio 7

pred caminoLibre (t: *tablero*, $p_0 : pos, p_1 : pos$){

$((\exists s : seq\langle pos\rangle)(s[0]_0 == p_0 \wedge s[|s|-1]_0 == p_1) \wedge (|s| \geq 3) \wedge$
$conMinasAdyacentes(p_1, t) \wedge PosicionValida(s[|s|-1]) \wedge$
$((\forall i : \mathbb{Z})(0 \leq i < |j| - 1)) \longrightarrow_L PosicionValida(s[i]) \wedge$
$PosicionesAdyacentes(s[i], s[i+1]) \wedge sinMinasAdyacentes(s[i], t))$

}

Hay un camino entre p0 y p1 tal que ninguna posicion intermedia del camino tiene minas alrededor. Dadas dos posiciones y un tablero, en este predicado nos aseguramos que haya una secuencia de posiciones que inicie en p0 y termine en p1, que todas las posiciones de esta secuencia sean adyacentes entre si, que todas las posiciones sean validas, y por ultimo, conMinasAdyacentes verifica que la cabeza de la secuencia tenga una mina adyacente, y con sinMinasAdyacentes nos fijamos que todas las demas no tengan.

3.8. Ejercicio 8

Igual que el procedimiento jugar pero descubriendo automaticamente los casilleros sin minas adyacentes.

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

Pre $\{j = j_0 \wedge (tableroValido(t) \wedge posValida(p, t, j_0)) \wedge_L (juegoValido(j_0, t)$
$\wedge casilleroNoDescubierto(j_0, p) \wedge casilleroSinBandera(b, p) \wedge \neg minaPisada(j_0, t))\}$

Post $\{(p \in j) \wedge (\forall i : \mathbb{Z})(0 \leq i < |j_0|) \longrightarrow_L$
$(j_0[i] \in j) \wedge (\forall x : pos)(caminoLibre(t, p, x)) \vee (caminoLibre2(t, p, x)) \longrightarrow_L (x \in j) \wedge$
$ningunoDeMas(t, p, j, j_0)\}$

}

Como precondicion pedimos tablero valido, pos valida, jugadas validas, casillero no descubierto y casillero sin bandera. Como postcondicion, nos aseguramos que la jugada que vamos a hacer sea parte del out, que tambien lo sean todas las que estan en el camino libre (con caminoLibre2 agregamos las jugadas que estan entre la posicion inicial y la ultima del camino libre.) y con ningunoDeMas nos aseguramos que los elementos del out sean solamente de la secuencia de jugadas iniciales, o la posicion jugada, o de los caminos libres.

3.9. Ejercicio 9

Dado un estado de juego con un patron 121, devuelve en p una posicion tal que, si se la juega, no se va a perder el juego.

proc sugerirAutomático121 (In t:*tablero*, In b:*banderitas*, In j:*jugadas*, Out p:*pos*) {

Pre $\{(tableroValido(t) \wedge juegoValido(t, j)) \wedge_L es121(t, j) \wedge \neg minaPisada(j, t)\}$

Post $\{(IzqODerDe121(t, j, p) \ \vee \ ArribaOAbajoDe121(t, j, p)) \ \wedge$
$(CasilleroSinBandera(b, p) \ \wedge \ CasilleroNoDescubierto(j, p))\}$

}

Como precondicion chequeamos que se cumpla el patron 121, ya sea horizontal o vertical. Como postcondicion, nos fijamos que si hay un patron 121 horizontal, y alguna posicion de las que esta arriba o abajo del casillero con 2 minas adyacentes esta jugada, entonces devuelve la opuesta (por ejemplo, si esta jugado arriba, devuelve como pos a jugar la de abajo). Analogamente, si el 121 esta en forma vertical, devuelve la posicion izquierda o derecha.

4. Predicados y funciones auxiliares

Creadas Para Ejercicio 2

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

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

  }

Dado un tablero y una secuencia de posiciones jugadas, el predicado verifica si toda posición que pertenece a dicha secuencia está en el rango del tablero y, a su vez, que las minas adyacentes en cada elemento de la secuencia de jugadas se corresponda con las minas adyacentes de cada posicion en el tablero.

4.1. Creadas Para Ejercicio 3

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

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

  }

Tomando como entrada una posición y un tablero (donde esa posición existe), para que sea verdadero el predicado debe ocurrir que la posición no está incluida en las jugadas, es decir, que no existe ningún elemento en la secuencia de jugadas tal que ese elemento es igual a la posición que se evalúa.

- `pred CasilleroSinBandera` (b:*banderitas*, p:*pos*) {

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

  }

Tomando como entrada una posición y una secuencia de posiciones en donde el jugador puso banderitas, para que sea verdadero el predicado debe ocurrir que la posición no está incluida en la secuencia de banderitas, es decir, que no existe ningún elemento en la secuencia de banderitas tal que ese elemento es igual a la posición que se evalúa.

- `pred esMatriz` (m: $seq\langle seq\langle \mathbb{Z} \rangle\rangle$) {

  $(\forall i : \mathbb{Z})((0 \leq i < filas(m)) \longrightarrow_L |m[i]| > 0 \wedge$
  $(\forall j : \mathbb{Z})((0 \leq j \leq filas(m)) \longrightarrow_L |m[i]| = |m[j]|)$

  }

Este predicado fue visto en una de las clases teóricas de la materia. Sirve para evaluar si es o no es Matriz, una secuencia de secuencia de enteros.

- `aux filas` (m: $seq\langle seq\langle \mathbb{Z} \rangle\rangle$) : $\mathbb{Z} = |m|$;

Este auxiliar fue visto en una de las clases teóricas de la materia. Cuenta la cantidad de filas que hay en una secuencia de secuencia de enteros.

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

Este auxiliar fue visto en una de las clases teóricas de la materia. Cuenta la cantidad de columnas que hay en una secuencia de secuencia de enteros.

- `pred tableroValido` (t: $seq\langle seq\langle \mathbb{Z} \rangle\rangle$) {
  $esMatriz(t) \wedge filas(t) = columnas(t)$
  }

Para ver si un tablero es un tablero válido evalúa si la secuencia de secuencia de enteros es una matriz, y a su vez hay la misma cantidad de filas que de columnas.

- `pred minaPisada` (j: *jugadas*, t:*tablero*) {
  $(\exists k : \mathbb{Z})(0 \leq k < |j| \wedge_L t[j([k]_{0_0}][(j[k]_{0_1}] = True)$
  }

Dada una secuencia de posiciones jugadas y un tablero, el predicado es verdadero si existe al menos una posicion incluida en la secuencia de posiciones jugadas y en el tablero que tenga una mina.

4.2. Creadas Para Ejercicio 5

- `pred todasDescubiertas` (j: *jugadas*, t:*tablero*) {
  $((\forall i, k : \mathbb{Z})(0 \leq i, k < |t| \wedge_L$
  $t[i][k] = False)) \longrightarrow$
  $(\exists n : pos \times \mathbb{Z})(n \in k \wedge_L t[n_{0_0}][n_{0_1}] = t[i][k])$
  }

Dada una secuencia de posiciones jugadas y un tablero, para toda posicion que no tiene una mina en el tablero, entonces implica que dicha posicion este incluida en la secuencia de posiciones de jugadas

4.3. Creadas Para Ejercicio 6

- `pred posValida` (p:*pos*, t:*tablero*, j: *jugadas*) {
  $(\forall i : \mathbb{Z})(0 \leq i < |j|) \longrightarrow_L$
  $\neg(j[i]_0 = p) \wedge (0 \leq p_0, p_1 < |t|)$
  }

Llamamos PosValida a una posicion que pertenece al tablero dado y que no esta descubierta, es decir que no esta incluida en la secuencia de posiciones jugadas.

4.4. Creadas Para Ejercicio 7

- `pred PosicionesAdyacentes` ($p_0 : pos, p_1 : pos$){

$$(-1 \leq (p_{0_0} - p_{1_0}) \leq 1) \wedge (-1 \leq (p_{0_1} - p_{1_1}) \leq 1)$$

}

El predicado es verdadero si y solamente si las posiciones son adyacentes una con la otra

`pred sinMinasAdyacentes` (p:*pos*, t:*tablero*) {

$(minasAdyacentes(t, p) == 0)$

}

El predicado es verdadero si y solamente si dado un tablero y una posición, la posición no tiene minas adyacentes en dicho tablero.

`pred conMinasAdyacentes` (p:*pos*, t:*tablero*) {

$(minasAdyacentes(t, p) > 0)$

}

El predicado es verdadero si y solamente si dado un tablero y una posición, la posición tiene minas adyacentes en dicho tablero.

4.5. Creadas Para Ejercicio 8

- `pred caminoLibre2` (t: *tablero*, $p_0 : pos, p_1 : pos$){

$$\begin{array}{l}(\exists s : seq\langle pos\rangle)(s[0] == p_0 \wedge s[|s| - 1] == p_1) \wedge (|s| > 1) \wedge \\ sinMinasAdyacentes(p_1, t) \wedge PosicionValida(s[|s| - 1]) \wedge \\ (\forall i : \mathbb{Z})(0 \leq i < |s| - 1) \longrightarrow_L PosicionValida(s[i]) \wedge \\ PosicionesAdyacentes(S[i], S[i + 1]) \wedge sinMinasAdyacentes(S[i], t)\end{array}$$

}

En este predicado vemos si existe una secuencia de posiciones tal que la primera posición es una de las posiciones dadas, y la ultima de la secuencia es la otra posición dada. Luego nos aseguramos que en toda la secuencia sean posiciones validas, todas las posiciones de la secuencia sean adyacentes entre sí y que todas las posiciones no tengan minas adyacentes. Este predicado va a dar verdadero si existe un camino entre esos dos puntos y que en todo ese camino no haya minas adyacentes.

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

$$\begin{array}{l}(\forall q : pos \times \mathbb{Z})(q \in j \longrightarrow_L (q_0 = p) \vee q \in j_0 \vee \\ caminoLibre(t, p, q_0) \vee caminoLibre2(t, p, q_0))\end{array}$$

}

Chequeamos que toda posicion que pertenezca a jugadas sea o bien p, o pertenezca en la entrada de jugadas, o sea parte de los caminos libres que se forman, es decir que no haya posiciones de mas.

4.6. Creadas Para Ejercicio 9

- `pred es121Horizontal` (j: *jugadas*, t:*tablero*) {

  $(\exists s : seq\langle pos \times \mathbb{Z}\rangle) \land cumple121Horizontal(j, t, s)$

  }

- `pred cumple121Horizontal` (j: *jugadas*, t:*tablero*, s:$seq\langle pos \times \mathbb{Z}\rangle$) {

  $(|s| = 3 \land (\forall i : \mathbb{Z})(0 \leq i < |s| \longrightarrow_L s[i] \in j) \land$
  $(s[0]_{0_0} = s[1]_{0_0} = s[2]_{0_0}) \land$
  $(s[0]_{0_1} = s[1]_{0_1} - 1 = s[2]_{0_1} - 2) \land$
  $(s[0]_1 = 1) \land (s[1]_1 = 2) \land (s[2]_1 = 1)$

  }

Tomando como entrada el tablero y las jugadas, nos fijamos si existe una secuencia de 3 jugadas que pertenezcan a j, y que cumplan con el patron 121 en manera horizontal. Para ello, nos fijamos en la primer jugada de la secuencia, con el primer subindice hacemos referencia a la tupla posicion, y con el segundo a la coordenada. En la segunda linea del predicado chequeamos que todas pertenezcan a la misma fila, en la tercera linea nos fijamos que esten una al lado de la otra, y en la ultima nos referimos con subindice 1 a la cantidad de minas adyacentes de la posicion. De esta forma, chequeamos que cumpla el patron 121.

- `pred es121Vertical` (j: *jugadas*, t:*tablero*) {

  $(\exists s : seq\langle pos \times \mathbb{Z}\rangle) \land cumple121Vertical(j, t, s)$

  }

- `pred cumple121Vertical` (j: *jugadas*, t:*tablero*, s:$seq\langle pos \times \mathbb{Z}\rangle$) {

  $(|s| = 3 \land (\forall i : \mathbb{Z})(0 \leq i < |s| \longrightarrow_L s[i] \in j) \land$
  $(s[0]_{0_1} = s[1]_{0_1} = s[2]_{0_1}) \land$
  $(s[0]_{0_0} = s[1]_{0_0} - 1 = s[2]_{0_0} - 2) \land$
  $(s[0]_1 = 1) \land (s[1]_1 = 2) \land (s[2]_1 = 1)$

  }

Analogamente al punto anterior, chequeamos que se cumpla el patron pero invertimos filas con columnas.

- `pred es121` (t:*tablero*, j: *jugadas*) {

  $(es121Vertical(t, j)) \lor (es121Horizontal(t, j))$

  }

Junta los dos predicados anteriores.

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

  $(\exists s : seq\langle pos \times \mathbb{Z}\rangle) \land (cumple121Vertical(t, j, s) \land$
  $((\exists r : pos)(s[1]_{0_0} = r_0 \land s[1]_{0_1} + 1 = r_1) \land_L$
  $(\exists n : \mathbb{Z})(0 \leq n < |j| \land j[n]_0 = r) \longrightarrow (p_0 = s[1]_{0_0} \land p_1 = s[1]_{0_1} - 1)) \lor$
  $((s[1]_{0_0} = r_0 \land s[1]_{0_1} - 1 = r_1) \land_L$
  $(\exists n : \mathbb{Z})(0 \leq n < |j| \land j[n]_0 = r) \longrightarrow (p_0 = s[1]_{0_0} \land p_1 = s[1]_{0_1} + 1)))$

  }

Nos fijamos que sea 121 vertical. Luego, nos fijamos en las posiciones izquierda y derecha. Si existe una posicion r que este a la derecha de la posicion con dos minas adyacentes y que este jugada, nos devuelve nos garantiza que la posicion p que vamos a devolver este a la izquierda del 2 (y que todavia no este jugada). Luego, ponemos un (o) para hacer lo mismo pero invirtiendo los lados.

- `pred ArribaOAbajoDe121` (t:*tablero*, j: *jugadas*, p:*pos*) {

  $(\exists s : seq\langle pos \times \mathbb{Z}\rangle) \wedge (cumple121Horizontal(t, j, s) \wedge$
  $((\exists r : pos)(s[1]_{0_0} - 1 = r_0 \wedge s[1]_{0_1} = r_1) \wedge_L$
  $(\exists n : \mathbb{Z})(0 \leq n < |j| \wedge j[n]_0 = r) \longrightarrow (p_0 = s[1]_{0_0} + 1 \wedge p_1 = s[1]_{0_1})) \vee$
  $((s[1]_{0_0} + 1 = r_0 \wedge s[1]_{0_1} = r_1) \wedge_L$
  $(\exists n : \mathbb{Z})(0 \leq n < |j| \wedge j[n]_0 = r) \longrightarrow (p_0 = s[1]_{0_0} - 1 \wedge p_1 = s[1]_{0_1})))$

  }

  Analogo a esIzquierdaODerecha, pero para el caso horizontal.