

Algoritmos y Estructuras de Datos I

Taller de matrices y tableros

Este taller contiene una serie de ejercicios sobre matrices y tableros para resolver y en el archivo `template-alumnos.zip` encontrarán una serie de casos de test, como habíamos visto en laboratorios pasados. Como parte del ejercicio de este labo, deben convertir los casos de test propuestos en el archivo `cases.cpp` a su versión GTEST como vimos en el laboratorio pasado. Se incluye en `template-alumnos.zip` el folder `lib` con GTEST.

Para este laboratorio, además deberán crear el `CMakeList.txt` en la carpeta que es abierta como proyecto de CLion (pueden basarse en el proyecto del laboratorio pasado como ejemplo). No olvidar modificar los archivos fuente (`SOURCE_FILES` y el nombre del proyecto)). Para compilar con GTEST utilicen como base el del labo correspondiente.

No olvidarse de descomprimir la librería GTEST dentro del folder `lib`.

Ejercicio 1: Dados dos vectores, calcular la matriz que resulta de hacer el producto vectorial entre ambos.

```
proc productoVectorial (in u: seq<Z>, in v: seq<Z>, out res: seq<seq<Z>>)) {
  Pre {True}
  Post {esMatrizDeAltoYAncho(res, |u|, |v|)  $\wedge_L$  cadaCoordenadaEsElProducto(res, u, v)}
}

pred esMatrizDeAltoYAncho (mat: seq<seq<Z>>, alto: Z, ancho: Z) {
  |mat| = alto  $\wedge$  todasLasFilasTienenAncho(mat, ancho)
}

pred todasLasFilasTienenAncho (matriz: seq<seq<Z>>, ancho: Z) {
  ( $\forall i : Z$ )( $0 \leq i < |matriz| \rightarrow_L |matriz[i]| = ancho$ )
}

pred cadaCoordenadaEsElProducto (producto: seq<seq<Z>>, vectorFila: seq<Z>, vectorColumna: seq<Z>) {
  ( $\forall i : Z$ )( $\forall j : Z$ )( $0 \leq i < |vectorFila| \wedge_L 0 \leq j < |vectorColumna| \rightarrow_L producto[i][j] = vectorFila[i] * vectorColumna[j]$ )
}
```

Ejercicio 2: Dada una matriz cuadrada, modificarla para obtener su traspuesta.

```
proc traspone (inout m: seq<seq<Z>>)) {
  Pre {m = m0  $\wedge$  esCuadrada(m)}
  Post {esMatrizDeAltoYAncho(m, |m0|, |m0|)  $\wedge_L$  cadaCoordenadaEsLaTraspuesta(m, m0)}
}

pred esCuadrada (m: seq<seq<Z>>)) {
  ( $\forall i : Z$ )( $0 \leq i < |m| \rightarrow_L |m[i]| = |m|$ )
}

pred cadaCoordenadaEsLaTraspuesta (traspuesta: seq<seq<Z>>, original: seq<seq<Z>>)) {
  ( $\forall i : Z$ )( $\forall j : Z$ )( $0 \leq i < |traspuesta| \wedge 0 \leq j < |traspuesta| \rightarrow_L traspuesta[i][j] = original[j][i]$ )
}
```

Ejercicio 3: Multiplicar matrices.

```
proc multiplicar (in m1: seq<seq<Z>>, in m2: seq<seq<Z>>, out res: seq<seq<Z>>)) {
  Pre { |m1| > 0  $\wedge$  |m2| > 0  $\wedge_L$  |m2[0]| > 0  $\wedge$  |m1[0]| = |m2|  $\wedge_L$ 
  todasLasFilasTienenAncho(m1, |m1[0]|)  $\wedge$  todasLasFilasTienenAncho(m2, |m2[0]|) }
  Post {esMatrizDeAltoYAncho(res, |m1|, |m2[0]|)  $\wedge_L$  ( $\forall i : Z$ )( $\forall j : Z$ )( $0 \leq i < |m1| \wedge_L 0 \leq j < |m2[0]| \rightarrow_L res[i][j] =$ 
   $\sum_{k=0}^{|m2|-1} m1[i][k] * m2[k][j]$ ) }
}
```

Ejercicio 4: Dada una matriz, devolver otra matriz reemplazando cada casillero por el promedio de la región compuesta por sus vecinos más el valor del centro.

```
proc promediar (in m: seq<seq<Z>>, out res: seq<seq<Z>>)) {
  Pre { |m|  $\geq 2 \wedge_L$  |m[0]|  $\geq 2 \wedge_L$  todasLasFilasTienenAncho(m, |m[0]|) }
  Post {esMatrizDeAltoYAncho(res, |m|, |m[0]|)  $\wedge_L$  cadaCoordenadaEsElPromedioDeLaRegion(res, m) }
}

pred cadaCoordenadaEsElPromedioDeLaRegion (res: seq<seq<Z>>, m: seq<seq<Z>>)) {
  ( $\forall i : Z$ )( $\forall j : Z$ ) ( $0 \leq i < |res| \wedge 0 \leq j < |res[i]| \rightarrow_L res[i][j] = promedioVecinos(m, i, j)$ )
}
```

```

}
aux promedioVecinos (m : seq⟨seq⟨ℤ⟩⟩, i: ℤ, j: ℤ) : ℤ = sumaVecinos(m, i, j) div cantidadVecinos(m, i, j);
aux sumaVecinos (m: seq⟨seq⟨ℤ⟩⟩, i: ℤ, j: ℤ) : ℤ =  $\sum_{a=i-1}^{i+1} \sum_{b=j-1}^{j+1}$  if vecinosEnRango(m, a, b) then m[a][b] else 0 fi;
aux cantidadVecinos (m: seq⟨seq⟨ℤ⟩⟩, i: ℤ, j: ℤ) : ℤ =  $\sum_{a=i-1}^{i+1} \sum_{b=j-1}^{j+1}$  if vecinosEnRango(m, a, b) then 1 else 0 fi;
pred vecinosEnRango (m: seq⟨seq⟨ℤ⟩⟩, i: ℤ, j: ℤ) {
  0 ≤ i < |m| ∧ 0 ≤ j < |m[a]|
}

```

Ejercicio 5: Contar cuántos picos tiene una matriz, donde un pico es un elemento que es mayor que todos sus vecinos.

```

proc contarPicos (in m: seq⟨seq⟨ℤ⟩⟩, out res: ℤ) {
  Pre { |m| ≥ 2 ∧L |m[0]| ≥ 2 ∧ todasLasFilasTienenAncho(m, |m[0]|) }
  Post { res =  $\sum_{i=0}^{|m|} \sum_{j=0}^{|m[i]|}$  if esPico(m, i, j) then 1 else 0 fi }
}
pred esPico (m : seq⟨seq⟨ℤ⟩⟩, i: ℤ, j: ℤ) {
  (∀a : ℤ)(∀b : ℤ)(esVecino(m, i, j, a, b) →L m[i][j] > m[a][b])
}
pred esVecino (m : seq⟨seq⟨ℤ⟩⟩, i: ℤ, j: ℤ, a: ℤ, b: ℤ) {
  (a ≠ i ∨ b ≠ j) ∧ i - 1 ≤ a ≤ i + 1 ∧ j - 1 ≤ b ≤ j + 1 ∧ enRango(m, a, b)
}

```

Ejercicio 6: Dada una matriz cuadrada, decidir si es triangular (inferior o superior).

```

proc esTriangular (in m: seq⟨seq⟨ℤ⟩⟩, out res: Bool) {
  Pre { esCuadrada(m) }
  Post { res = true ↔ esTriangularSuperior(m) ∨ esTriangularInferior(m) }
}
pred esTriangularInferior (m: seq⟨seq⟨ℤ⟩⟩) {
  (∀i : ℤ)(∀j : ℤ)(enRango(m, i, j) ∧ i < j →L m[i][j] == 0)
}
pred esTriangularSuperior (m: seq⟨seq⟨ℤ⟩⟩) {
  (∀i : ℤ)(∀j : ℤ)(enRango(m, i, j) ∧ j < i →L m[i][j] == 0)
}

```

Ejercicio 7: Decidir si, dado un tablero (no necesariamente de 8 x 8) con reinas de ajedrez, existen dos reinas que se amenazan entre sí.

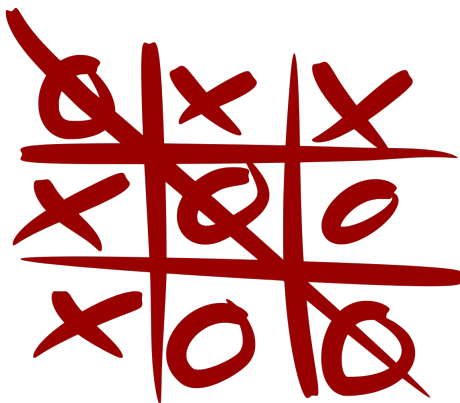
```

proc hayAmenaza (in m: seq⟨seq⟨ℤ⟩⟩, out res: Bool) {
  Pre { |m| ≥ 2 ∧L |m[0]| ≥ 2 ∧L todasLasFilasTienenAncho(m, |m[0]|) ∧ esBinaria(m) }
  Post { res = true ↔ existeAmenaza(m) }
}
pred esBinaria (m: seq⟨seq⟨ℤ⟩⟩) {
  (∀i : ℤ)(∀j : ℤ)(0 ≤ i < |m| ∧ 0 ≤ j < |m[i]| →L 0 ≤ m[i][j] ≤ 1)
}
pred existeAmenaza (m: seq⟨seq⟨ℤ⟩⟩) {
  (∃i1 : ℤ)(0 ≤ i1 < |m| ∧L (∃j1 : ℤ)(0 ≤ j1 < |m[i1]| ∧L m[i1][j1] = 1 ∧ amenazaAlguna(m, i1, j1)))
}
pred amenazaAlguna (m: seq⟨seq⟨ℤ⟩⟩, i1: ℤ, j1: ℤ) {
  (∃i2 : ℤ)(0 ≤ i2 < |m| ∧L (∃j2 : ℤ)(0 ≤ j2 < |m[i2]| ∧L m[i2][j2] = 1 ∧ seAmenazan(i1, j1, i2, j2)))
}
pred seAmenazan (i1: ℤ, j1: ℤ, i2: ℤ, j2: ℤ) {
  (i1 ≠ i2 ∨ j1 ≠ j2) ∧ (i1 = i2 ∨ j1 = j2 ∨ abs(i1 - i2) = abs(j1 - j2))
}
aux abs (t : ℤ) : ℤ = if t ≥ 0 then t else -t fi;

```

Ejercicio 8: Dada una matriz cuadrada de $n \times n$, devolver la diferencia absoluta entre la suma de sus dos diagonales. Una diagonal es la que empieza en la posición (0,0) y termina en (n-1,n-1), y la otra que va entre las posiciones (0,n-1) y (n-1,0).

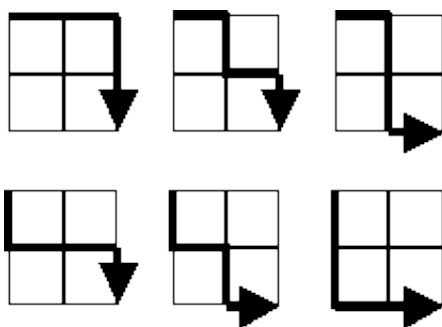
Ejercicio Adicional TaTeTi: Escribir un algoritmo que verifique si una partida de TaTeTi está terminada.



Muy fácil? Ahora generalizarlo para un tateti de N columnas y N filas.

Generar varios TESTs para verificar la implementación.

Ejercicio Adicional "Willy, el robot" Supongamos que tenemos un robot sentado en la esquina arriba izquierda de una grilla de $X \times Y$. El robot se puede mover en dos direcciones: para abajo y para la derecha.



Escribir un algoritmo que determine cuántos caminos posibles puede hacer el robot para llegar de la posición $(0,0)$ a la (X,Y) . Queda prohibido usar la fórmula cerrada para calcularlo.

Generar varios TESTs para verificar la implementación.