Pregunta 1 (15 puntos) - Iteración en matrices

Se desea implementar un programa de "Sopa de Números" , utilizando una Matriz S de dimensiones $[0..N) \times [0..M)$, de números enteros entre 0 y 9.

Debido a la magnitud del problema, por los momentos nos enfocaremos en resolver el subproblema de buscar un número positivo en la matriz, en dirección diagonal descendente, de izquierda a derecha. El número a buscar estará almacenado en un arreglo de tamaño T. Cada posición del arreglo almacenará un dígito del número; el número tendrá T dígitos.

Esta versión del pasatiempo tiene la particularidad de que el número a buscar puede aparecer varias veces en la matriz. Se desea encontrar todas las ocurrencias del número. Por ejemplo, en la figura 1 se muestra una matriz 5×6 con 3 ocurrencias del número B=[2, 0, 2, 1] en dirección diagonal descendente, de izquierda a derecha.

$$S = \begin{bmatrix} 4 & 2 & 5 & 2 & 2 & 3 \\ 2 & 2 & 0 & 6 & 3 & 6 \\ 2 & 0 & 0 & 2 & 2 & 7 \\ 1 & 8 & 2 & 2 & 1 & 8 \\ 0 & 9 & 0 & 1 & 1 & 9 \end{bmatrix}$$

Figura 1: Ejemplo de ocurrencias del número 2021 en dirección diagonal descendente, de izquierda a derecha, en la matriz S

- Programa correctamente en GCL la **función**:

  ddl_from_pos(i, j, N, M, T: int, S: array [0..N)× [0.. M) of int, B: array [0..T) of int) -> bool

  que busca en la matriz el número de T dígitos almacenado en el arreglo B. La búsqueda se realiza comenzando en la posición i, j de la matriz S y de manera diagonal descendente, de izquierda a derecha. La función retorna true si el número fue hallado en la matriz, y retorna false en caso contrario. Esta función no impone precondiciones sobre el tamaño máximo de T (la función retorna false si el número no se encuentra en la matriz). Indica la postcondición, invariantes y cotas. No indiques la precondición.

- Programa correctamente en GCL la **función**: **ddl_occurrences( )**

  que retorna la cantidad de ocurrencias de un número dado, en dirección diagonal descendente de izquierda a derecha en una matriz. Esta función utiliza a la función **ddl_from_pos( )**. Incluye la definición (firma completa) de la función y la cota de la iteración. No indiques las demás aserciones.

- **Diseña la solución general**:

  explica todo lo que haga falta programar en GCL para tener una implementación completa de la "Sopa de Números" con ocurrencias mútiples, de varios números de distintas cantidades de dígitos postitivos. No programes estas funciones y procedimientos, ni indiques su firma completa, pero explica el diseño de la solución, los problemas importantes a resolver y cómo se resolverían, así como la estructura del algoritmo que integraría a todos estos subprogramas.

Nota que cualquier pregunta puede responderse aunque no se hayan respondido las demás.

Sea

con $c : array[0..5)$ of int;

donde las posiciones del arreglo c contienen los últimos 5 dígitos de tu carné estudiantil (tu carné sin los dígitos referentes a la cohorte).

Por ejemplo, para un estudiante cohorte 20, cuyo carné es 2021034, se tiene que:

$$c[0] = 2,\ c[1] = 1,\ c[2] = 0,\ c[3] = 3,\ c[4] = 4.$$

Dada la siguiente tripleta de Hoare:

$$\{b[2] = c[2]\} \quad b[b[2] \, mod \, 2] := c[4] \quad ; \quad b[c[0]] := c[0] \quad \{b[b[1]] \leq b[2]\}$$

Si la tripleta puede ser verdadera, demuéstralo formalmente. De ser necesarias, indica qué condiciones adicionales se requieren para que la tripleta sea verdadera, y explica detalladamente por qué.

En cambio, si la tripleta es falsa, muéstralo y justifica detalladamente por qué.

Incluye en tu respuesta el número de tu carné.

Esta es una pregunta sencilla, pero **se evaluará rigurosamente, por lo que sugiero prestar cuidado a la sintaxis de las expresiones así como a los argumentos y procedimientos presentados**.

Pizza Party (8 puntos)

El teorema de los números poligonales centrados, mejor conocido como el teorema del cortador perezoso (lazy caterer), describe la cantidad máxima de pedazos de un círculo que pueden lograrse con un número dado de cortes rectos (no necesariamente todos los cortes tiene el mismo punto en común). Ver animación:

https://commons.wikimedia.org/wiki/File:Lazy_Caterer%27s_Sequence_(Cuts).gif

Por ejemplo, si no se realiza ningún corte se tiene un solo pedazo (el círculo completo), mientras que un corte produce dos pedazo, dos cortes producen cuatro pedazos, tres cortes pueden producir hasta siete pedazos, y así sucesivamente.

El teorema establece que el número máximo de pedazos p que se pueden producir con n cortes es:

$$p = \frac{n^2+n+2}{2}$$

Se desea utilizar el teorema del cortador perezoso realizar cortes en Z pizzas, para compartir los pedazos entre un grupo de H humanos (H>=0), s uponiendo que la persona que realiza los cortes lo hace maximizando el número de pedazos a obtener, según el teorema dado.

Para hacer más sencilla la implementación, en vez de utilizar múltiples guardias en la instrucción de iteración de GCL, se utilizará una función llamada random, similar a la discutida en clase (https://www.dc.fi.udc.es/~cabalar/gcl/random.gcl) pero que dado un número entero N, retornará un número entero cualquiera entre 0 y N. Esta función se utilizará para elegir la pizza a cortar en una iteración dada.

func random(N: int) -> int

{P: N >= 0}

{Q: 0<= func <= N}

- Implementa un **procedimiento** correcto en GCL para resolver el problema planteado. Para cada corte, el programa (Z-perezoso) elige indistintamente una de las pizzas. El proceso de cortes termina apenas haya suficientes pedazos de pizza para repartir entre todos los humanos del grupo (al menos un pedazo por humano). Nota que al inicio ya se tienen Z pedazos de pizza (cada pizza completa es un pedazo). Además, el número total de pedazos varía dependiendo de en qué pizza se realiza cada corte, por lo que se requiere mantener el número de cortes realizados en cada pizza, para así saber cuántos pedazos hay en total y poder detener la iteración. Este procedimiento debe utilizar la función random para elegir la pizza a cortar en cada iteración, de modo que la iteración tendrá un solo comando con guardia pero la ejecución seguirá siendo no determinista. Anota la precondición, postcondición, invariante y cota. La cota debe acercarse lo más posible a cero al finalizar la iteración. Por supuesto, todas las aserciones deben describir lo mejor posible el problema a resolver. Indica la definición (firma) del procedimiento: parámetros y su tipo, así como si son de entrada, salida o entrada y salida. Recuerda que las únicas operaciones aritméticas disponibles en GCL son +, - , *, div y mod.

```
stop:boolean;
i,stop:=0,false;
do not stop and i<>4 -> i:=i+1
 | not stop and i<>4 -> stop:=true
od
```