Resolución BufferCircular

Aclaraciones

El TAD ColaAcotada tiene dos observadores:

```
        obs cap: int      // Nos determina la cantidad de elementos que podemos tener en la Cola
        obs s:   seq<T>   // Contiene los elementos que están almacenados en la Cola
```

Por otro lado, vuelvo a aclarar lo que vimos en la clase. En la especificación tenemos que hablar del mundo de la especificación, por lo que vamos a "acceder" a los observadores de los módulos que compongan nuestra estructura. La excepción a esta regla son los tipos básicos (int, bool, etc).
Mientras tengamos elementos que son de la implementación, podemos utilizar sus operaciones.
De esta forma, podemos obtener la longitud del arreglo con la operación "length" sin necesidad de acceder al observador del TAD Lista.

La interpretación es que al encolar se introducen elemento en el índice `fin` y al desencolar se obtiene el elemento en la siguiente posición al índice `inicio`. Al encolar `fin` se desplaza una posición a la derecha, mientras que al desencolar `inicio` se desplaza una posición a la izquierda.

En el arreglo no interesa qué contienen las posiciones que aparecen en verde en el diagrama, ya que corresponden a posiciones que no están siendo utilizadas por el Buffer/Cola. Los elementos con fondo blando, que se ubican entre las posiciones marcadas por `inicio` y `fin`, deben estar en el mismo orden en que fueron encolados y deben coincidir con lo que se espera según el TAD.

Primero conviene observar qué ocurre cuando las posiciones verdes están en los extremos y cuando están en el medio del arreglo. Luego, pensemos qué ocurre cuando el buffer está lleno (todas posiciones ocupadas con valores) y cuando está vacío (todas verdes). Finalmente, extendamos el análisis con algunos casos más que creamos que pueden ser de interés (sólo posiciones verdes en la parte derecha, sólo en la parte izquierda, si el buffer tiene capacidad nula, capacidad de un elemento, de dos, etc.). Para cada caso no olviden pensar qué ocurre con las variables de nuestra estructura.

Análisis

Voy a representar los ejemplos con una V para la posición vacío o de color verde en el diagrama y con números para las que estén ocupadas. Usaré un buffer de tamaño 7 salvo que se indique lo contrario. Pondré las etiquetas i y f para referirme a inicio y fin.

Posiciones verdes en los extremos

```
   i
[V,V,1,2,3,V,V]
           f
```

inicio: 1, fin: 5

Posiciones verdes en el centro

```
         i
[3,4,V,V,V,1,2]
     f
```

inicio: 4, fin: 2
cant: 4

Buffer lleno

```
         i
[3,4,5,6,7,1,2]
           f
```

inicio: 4, fin: 5

Buffer vacío

```
         i
[V,V,V,V,V,V,V]
```

```
           f

inicio: 4, fin: 5

### Posiciones verdes al final

             i
[1,2,3,4,V,V,V]
         f

inicio: 6, fin: 4

### Posiciones verdes al principio

       i
[V,V,V,V,1,2,3]
 f

inicio: 3, fin: 0

### Buffer de tamaño 0

[]

inicio: ?, fin: ?

### Buffer de tamaño 1 vacío

 i
[V]
 f

inicio: 0, fin: 0

### Buffer de tamaño 1 lleno

 i
[1]
 f

inicio: 0, fin: 0

### Buffer de tamaño 2 vacío

   i
[V,V]
 f

inicio: 1, fin: 0

### Buffer de tamaño 2 con un elem

   i
[1,V]
   f

inicio: 1, fin: 1

### Buffer de tamaño 2 lleno

   i
[1,2]
 f

inicio: 1, fin: 0

### Conlusiones

- Parecen pasar cosas raras cuando el buffer no tiene espacio, sería un caso para excluir.
```

- Con las variables inicio, fin y el arreglo no puedo distinguir los casos cuando el buffer está lleno o vacío, no sé si tengo que comprobar los elementos o no. Para solventar esto podemos sumar una variable que nos diga cuántos elementos contiene el buffer.

- Existe una relación de desigualdad entre inicio y fin. Esa relación me permite separar en casos. Las excepciones a esto se encuentran cuando el arreglo está lleno o vacío, pero ambos casos pueden contenerse igualmente dentro de los dos casos que vamos a describir a continuación

    0. Si el buffer está vacío, lo consideramos como un caso particular ya que no hay elems.

    1. Si inicio es menor estricto que fin, estamos en el caso de que las posiciones que van desde inicio+1 hasta fin-1 son las de la Cola.

    2. Si inicio es mayor o igual que fin, estamos en el caso que tenemos los elementos de la Cola partidos en dos subarreglos, (A) uno que va desde inicio+1 hasta el final del arreglo y (B) otro que va desde el comienzo del arreglo hasta fin-1. Si el buffer está lleno se comporta de esta misma manera aunque inicio sea más chico que fin.

Resolución

```
Modulo BufferCircular<T> implementa ColaAcotada<T> {
        var elems: Array<T> // Arreglo de tamaño fijo que contiene los elementos del Buffer
Circular
        var inicio: int     // Posición adyacente al próximo elemento del Buffer
        var fin: int        // Posición donde se introducirá un nuevo elemento
        var cant: int       // Cantidad de elementos contenidos en el Buffer

        pred InvarianteRepresentación(i: BufferCircular<T>) {
                // Sobre elems
                0 < i.elems.length          AND

                // Sobre inicio
                EnRango(i.inicio, i.elems)    AND

                // Sobre fin
                EnRango(i.fin, i.elems)       AND

                // Sobre cant
                0 ≤ i.cant ≤ i.elems.length AND

                // Sobre la relación entre cant, inicio, fin y elems
                // Caso 1 + Lleno
                (i.inicio < i.fin) -> (i.cant = i.fin - i.inicio - 1 OR i.cant = i.elems.length)
AND

                // Caso 2
                (i.inicio ≥ i.fin) -> (i.cant = i.fin + i.elems.length - i.inicio - 1)
        }

        pred EnRango(pos: Z, a: Array<T>) {
                0 ≤ pos < a.length
        }

        pred Abstracción(i: BufferCircular<T>, t: ColaAcotada<T>) {
                (t.cap = i.elems.length AND (i.cant = |t.s|)) AND_L

                (
                        (i.cant = 0 -> t.s = <>) AND

                        (i.cant > 0 ->
                                // Para la abstracción separamos en dos casos
                                // Necesitamos decir qué ocurre con t.s y relacionarlo con nuestro
array

                                // Caso 1. Las posiciones vacías están en las puntas del array y
no en el medio
                                (i.fin < i.inicio AND i.cant != t.cap) ->
```

```
MismosElementosEntre(i.elems, t.s, i.inicio+1, i.fin-1) AND

                                // Caso 2. Las posiciones vacías están en el medio del array y no
en las puntas o
                                // el buffer está lleno
                                (i.inicio ≥ i.fin OR i.cant = t.cap) ->
                                        MismosElementosEntre(i.elems, subseq(t.s, 0,
i.elems.length-i.inicio-1), i.inicio+1, i.elems.length-1) AND
                                        MismosElementosEntre(i.elems, subseq(t.s, i.elems.length-
i.inicio-1, |t.s|), 0, i.fin-1)
                        )
                )
        }

        proc Encolar(inout i: BufferCircular<T>, in e: T) {
                i.elems[i.fin] = e;
                i.fin = posicionAjustada(i.fin + 1);
        }

        proc Desencolar(inout i: BufferCircular<T>): T {
                res := i.elems[posicionAjustada(i.inicio + 1)];
                i.inicio := posicionAjustada(i.inicio + 1);
                return res;
        }

        // Implemento un prox auxiliar que me permite estandarizar cómo calculo
        // las posiciones en caso de que se hayan ido del rango del array.
        // Esto es lo que simula el comportamiento circular del Buffer.
        proc posicionAjustada(in i: BufferCircular<T>, in pos: Z): Z {
                return (pos) mod i.elems.length;
        }
}


### Análisis de la Abstracción

Veamos qué ocurre con el caso 2 de la abstracción, que es el que presenta más complejidad. Para
eso
pensemos qué pasa para los dos "subcasos" que dijimos que componen este caso.

#### Buffer no lleno e `inicio ≥ fin`

         i
[3,4,V,V,V,1,2]
      f

inicio: 4, fin: 2
cant: 4
i.elems.length = 7
i.elems.length-i.inicio = 3
i.elems.length-i.inicio-1 = 2

t.s = [1,2,3,4]
subseq(t.s, 0, 2) = [1,2]
subseq(t.s, 2, 4) = [3,4]

#### Buffer lleno

               i
      [3,4,5,6,7,1,2]
                 f

inicio: 4, fin: 5
cant: 7
i.elems.length = 7
i.elems.length-i.inicio = 3
i.elems.length-i.inicio-1 = 2

t.s = [1,2,3,4,5,6,7]
```

```
  subseq(t.s, 0, 2) = [1,2]
  subseq(t.s, 2, 7) = [3,4,5,6,7]
```