Diseño con Arreglos

Algoritmos y Estructuras de Datos

11 de octubre de 2024

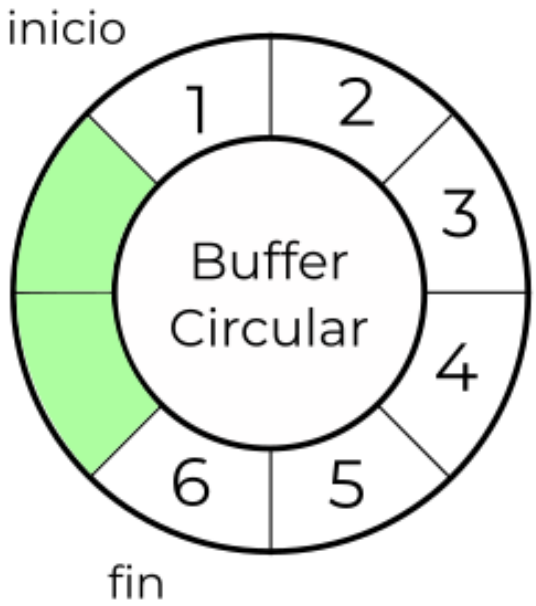
Plan hasta el final de clase

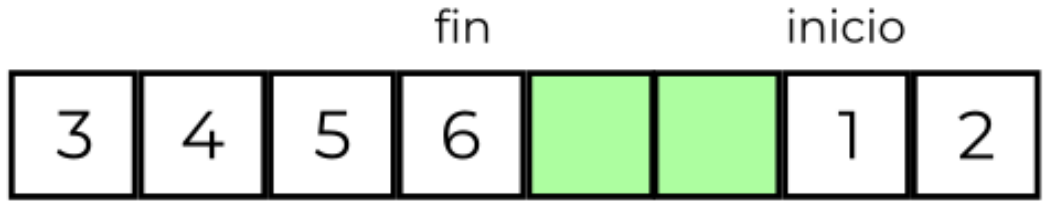
- Ejercicio de buffer circular
- Devolución de parciales al final

Práctica 6, ejercicio 5

Una forma eficiente de implementar el TAD Cola en su versión acotada (una cantidad máxima de elementos predefinida), es mediante un buffer circular. Esta estructura está formada por un array del tamaño máximo de la cola (n) y dos índices (inicio y f in), que indican en qué posición empieza y en qué posición termina la cola, respectivamente. Al encolar un elemento, se lo guarda en la posición indicada por el índice fin y se incrementa dicho índice. Al desencolar un elemento, se devuelve el elemento indicado por el índice inicio y se incrementa el mismo. En ambos casos, si el índice a incrementar supera el tamaño del array, se lo reinicia a 0.

Práctica 6, ejercicio 5





Nota: índice fin apunta a primera posición vacía

Lo primero, ¿qué TAD estamos implementando?

```
TAD Cola<T> {
    obs s: seq<T>

    proc colaVacía(): Cola<T>
        asegura {res.s = ⟨⟩}

    proc vacía(in c: Cola<T>): bool
        asegura {res = true ↔ c.s = ⟨⟩}

    proc encolar(inout c: Cola<T>, in e: T)
        requiere {c = C_0}
        asegura {c.s = concat(C_0.s, ⟨e⟩)}

    proc desencolar(inout c: Cola<T>): T
        requiere {c = C_0}
        requiere {c.s ≠ ⟨⟩}
        asegura {c.s = subseq(C_0.s, 1, |C_0.s|)}
        asegura {res = C_0[0]}

    proc proximo(in c: Cola<T>): T
        requiere {c = C_0}
        requiere {c.s ≠ ⟨⟩}
        asegura {res = C_0.s[0]} }
```

Vamos por partes dijo Jack

- Escriba el invariante de representación y la función de abstracción

Vamos por partes dijo Jack

- Escriba el invariante de representación y la función de abstracción

¿Ideas?

Para el invariante de representación, tenemos que relacionar las 3 variables de estado.

Repasemos enunciado: Esta estructura está formada por
un array del tamaño máximo de la cola (n) y dos índices (inicio y fin)

¿Entonces?

Llamando elems a la variable de estado que representa al array, nos queda:

InvRep(c': ColaAcotadaImpl):
$0 \leq c'.inicio < c'.elems.length \ \& \ \& \ 0 \leq c'.fin < c'.elems.length$

Analicemos ahora la función de abstracción

Tenemos como observador para el TAD cola a s, que representa a los elementos de la cola, y al estar acotada, tenemos otro observador de capacidad (llamémoslo cap), que nos dice cuántos elementos pueden entrar como máximo.

¿Cómo relacionamos a nuestras variables de estado con estos obs?

1) De alguna forma, hay que asociar tamaño del arreglo con la capacidad que tiene.

1) De alguna forma, hay que asociar tamaño del arreglo con la capacidad que tiene.

FuncAbs(c': ColaAcotadaImpl, c: ColaAcotada):
c.cap == c'.elems.length-1

¿Por qué es uno menos?

2) ¿Qué sucede con el tamaño de s en función de como se mueven inicio y fin?

Veamos en el gráfico

2) ¿Qué sucede con el tamaño de s en función de como se mueven inicio y fin?

Veamos en el gráfico

FuncAbs(c': ColaAcotadaImpl, c: ColaAcotada):
c.cap == c'.elems.length-1 & &
$(c'.fin - c'.inicio)$ mód $c'.elems.length == |c.s|$

3) Finalmente, ¿cómo referimos a los elementos en la cola?

3) Finalmente, ¿cómo referimos a los elementos en la cola?

FuncAbs(c': ColaAcotadaImpl, c: ColaAcotada):
c.cap == c'.elems.length-1 & &
$(c'.fin - c'.inicio)$ mód $c'.elems.length == |c.s|$ & &
$\forall i : \mathbb{Z} :: 0 \leq i < |c.s| \longrightarrow_L c.s[i] == c'.elems[(c'.inicio + i)$
mód $c'.elems.length]$

Ahora, los proc

- Escriba los algoritmos de las operaciones encolar y desencolar

¿Ideas?

Miremos sus especificicaciones. En principio, parecen ser similares, con la diferencia que en desencolar, devolvemos el elemento que estamos sacando.

```
proc encolar(inout c: Cola<T>, in e: T)
      requiere {c = C_0}
      asegura {c.s = concat(C_0.s, ⟨e⟩)}

proc desencolar(inout c: Cola<T>): T
      requiere {c = C_0}
      requiere {c.s ≠ ⟨⟩}
      asegura {c.s = subseq(C_0.s, 1, |C_0.s|)}
      asegura {res = C_0[0]}
```

¿Cómo usamos las variables de estado?

proc encolar(inout c': ColaAcotada, in e: T):

c'.elems[c'.fin] := e
c'.fin := (c'.fin + 1) mód $c'.elems.length$

¿Cómo hacemos desencolar?

proc desencolar(inout c': ColaAcotada): T

res := c'.elems[c'.inicio]
c'.inicio := (c'.inicio + 1) mód $c'.elems.length$
return
res

Finalmente

- ¿Por qué tiene sentido utilizar un buffer circular para una cola y no para una pila?

¿Ideas?