Estructuras de Datos

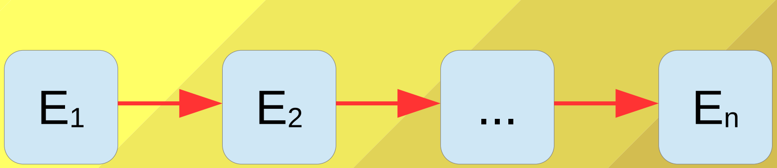
Estructuras lineales:
Arrays

Contenidos

- Características de la estructuras lineales.
- Arrays:
  - Array.
  - Array dinámico.
  - Array dinámico circular.

Introducción

- Estructuras lineales.
  - Contenedores de datos genéricos.
  - Relación 1-1: cada elemento tiene un predecesor y un sucesor (salvo el inicial y el final).
  - Indicadas cuando se realiza un proceso secuencial de los datos.



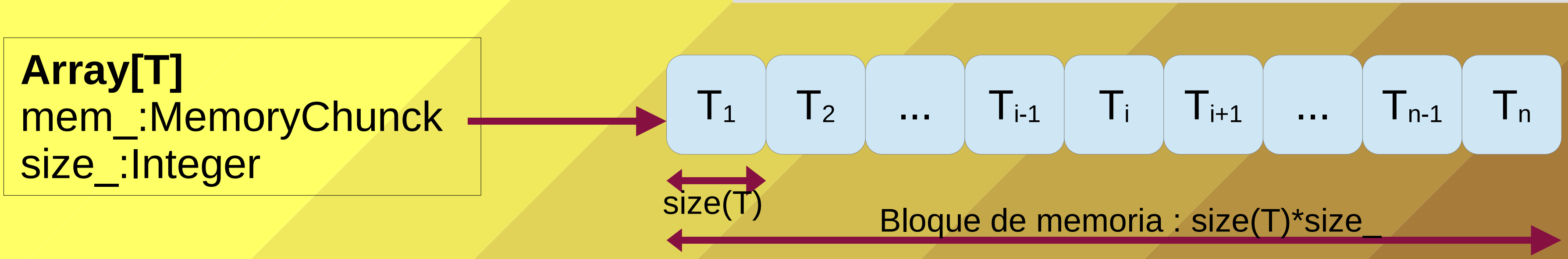
Arrays

- Arrays.

Array[T]:

- **Makers**:
  - make(size:Integer):Array[T]
    - *pre-c: size>0*
- **Observers**:
  - size():Integer
  - get(i:Integer)
    - *pre-c: i>=0 and i<size()*
- **Modifiers**:
  - set(i:Integer, item:T)
    - *pre-c: i>=0 and i<size()*
    - *post-c: get(i)==item*

```
Algorithm Array::make(size:Integer):Integer //O( 1 )
Begin
   mem_ <- getMemory(size(T)*size)
   size_ <- size
End.

Algorithm Array::size():Integer //O( 1 )
Begin
   Return size_
End.

Algorithm Array::get(i:Integer):T //O( 1 )
Begin
   Return convert[T](mem_+i*size(T))
End.

Algorithm DArray::set(i:Integer, v:T) //O( 1 )
Begin
   copy(memDir(v),memDir(v)+size(T), mem_+i*size(T))
End.
```



Array[T]
mem_:MemoryChunck
size_:Integer

Arrays

- Array dinámico.

Es un X que además ...

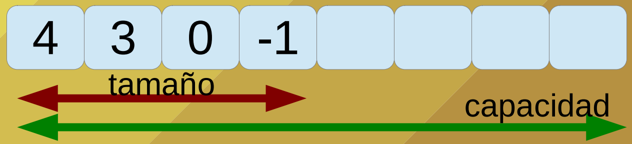
DArray[T] extends Array[T]:

- **Makers**:
  - **make():DArray**
    - *post-c: capacity()=1*
    - *post-c: size()=0*
- **Observers**:
  - **capacity():Integer**
  - **isEmpty():Bool**
    - *post-c: !**retV** || size()=0*
  - **isFull():Bool**
    - *post-c: !retV || capacity()=size()*
- **Modifiers**:
  - **pushBack(v:T)**
    - *post-c: size()=**old**.size()+1*
    - *post-c: get(size()-1)==v*
  - ***popBack()***
    - *prec-c: !isEmpty()*
    - *post-c: size()==old.size()-1*
    - *post-c: size()>=1 → get(size()-1)==old.get(old.size()-2)*

Valor retornado

El estado antes de realizar la operación.

- **Modifiers**:
  - **insert(i:Integer, v:T)**
    - *pre-c: 0<= i <size()*
    - *post-c: size()==old.size()+1*
    - *post-c: get(i)==v*
  - **remove(i)**
    - *pre-c: 0<= i <size()*
    - *post-c: size()==old.size()-1*
    - *post-c: i==size() or get(i)=old.get(i+1)*
- **Invariants:**
  - *size()<=capacity()*



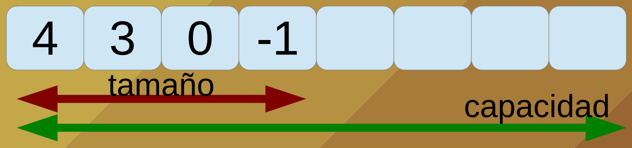
Arrays

- Array dinámico: diseño.

```
Algorithm DArray::isEmpty():Bool //O(1)
   Return size()=0

Algorithm DArray::isFull():Bool //O(n)
   Return size()=capacity()

Algorithm DArray::pushBack(v:T) //O( 1 )
Begin
   If (isFull()) Then
      grow(_data)
   data_.set(size_, v)
   size_ ← size_ + 1
End.

Algorithm DArray::popBack()//O(1)
Begin
   size_ ← size_ - 1
End.

Algorithm DArray::grow()//O(   )
Begin
   tmp ← Array(capacity()*2)
   copy[T](data_.mem_,0,size_,tmp.mem_)
   data_<- tmp
End.
```

```
Algorithm DArray::insert(i:Integer, v:T)//O( n )
Begin
   If (isFull()) then
      grow()
   For j ← size_-1 To i Inc -1 Do
      data_.set(j+1, data_.get(j))
   data_.set(i, v)
   size_ ← size_ + 1
End.

Algorithm DArray::remove(i:Integer)//O( n )
Begin
   For j <- i To _size-2 Inc 1 Do
      data_.set(j, data_.get(j+1))
   size_ ← size_ – 1
End.
```

DArray[T]:
data_:Array[T]
size_:Integer

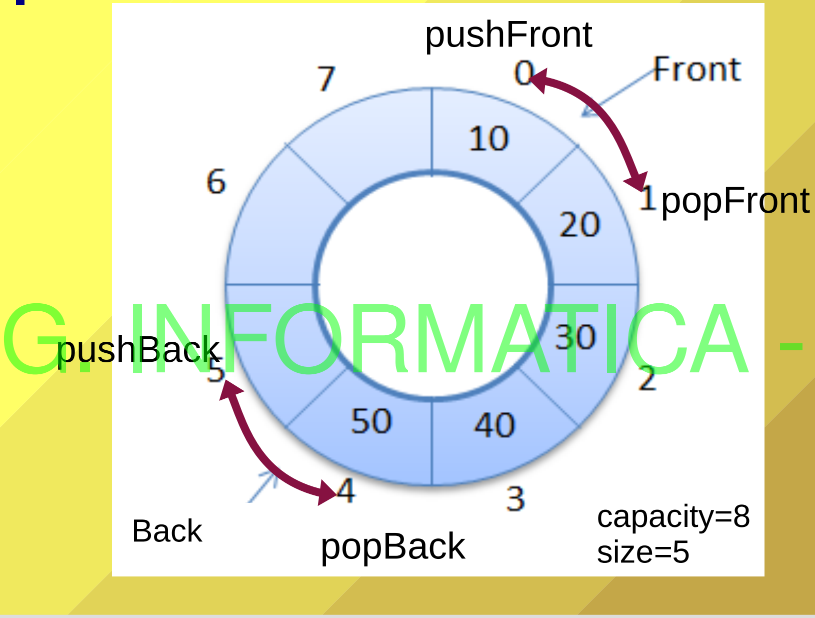


Arrays

- Array dinámico circular.

CDArray[T] extends DArray[T]:

- **Makers**:
  - **make():CDArray**
    - *post-c: capacity()=1*
    - *post-c: size()=0*
- **Observers**:
- **Modifiers**:
  - **pushFront(v:T)**
    - *post-c: size()=old.size()+1*
    - *post-c: get(0)=v*
  - ***popFront()***
    - *prec-c: !isEmpty()*
    - *post-c: size()=old.size()-1*
    - *post-c: isEmpty() || get(0)=old.get(1)*



```
Algorithm cInc(i:Integer, capacity:Integer):Integer
//O( 1 )
Begin
   Return (i+1)%capacity
End.

Algorithm cDec(i:Integer, capacity:Integer):Integer
//O( 1 )
Begin
   Return (i-1+capacity)%capacity
End.
```

Arrays

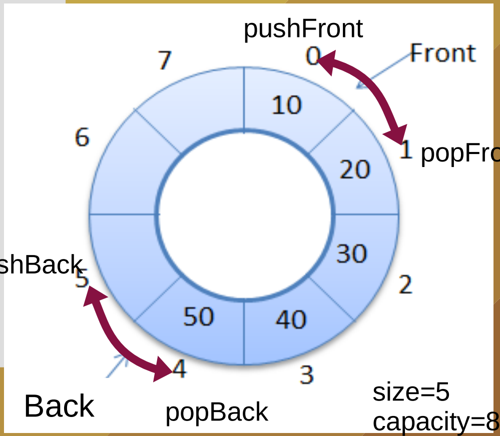
- Array dinámico circular.

```
Algorithm DArray::capacity():Integer
   Return data_.size()

Algorithm DArray::size():Integer
   Return size_

Algorithm DArray::pushBack(v:T)
//O( n ) CA( 1 )
Begin
   If isEmpty() Then
     front_<-0
     back_<- 0
   Else
     If (isFull()) Then
       grow()
     back_<-cInc(back_,capacity())
   End-If
   data_.set(back_,v)
   size_<-size_ + 1
End.

Algorithm DArray::popBack() //O(1)
Begin
   back_<-cDec(back_,capacity())
   size_<-size_ - 1
End.
```

```
Algorithm DArray::pushFront(v:T)
//O(n ) CA( 1 )
Begin
   If isEmpty() Then
     front_<-0
     back_<- 0
   Else
     If (isFull()) Then
       grow()
     front_<-cDec(front_,capacity())
   End-If
   data_.set(front_,v)
   size_<-size_ + 1
End.

Algorithm DArray::popFront() //O(1)
Begin
   front_<-cInc(front_,capacity())
   size_<-size_ - 1
End.
```

CDArray[T]

```
data_ :Array[T]
front_:Integer
back_ :Integer
size_ :Integer.
```



Arrays

- Array dinámico circular.

```
Algorithm DArray::get(i:Integer):T //O( 1 )CA(1)
Begin
   Return data_.get((front_+i)%capacity())
End.

Algorithm DArray::set(i:Integer, v:T) //O( 1 )
Begin
   data_.set((front_+i)%capacity(), v)
End.

Algorithm DArray::grow(data:Array[T]) //O( n )
Begin
   tmp <- Array(capacity()*2)
   For i <- 0 TO size()-1 Inc 1 Do
     tmp.set(i, get(i))
   front_ <- 0
   back_ <- size_-1
   data <- tmp
End.
```

```
Algorithm DArray::insert(pos:Integer, v:T)//O( n )
Begin
  If pos=0 Then
    push_front(v)
  Else
    If (isFull()) Then
      grow()
    size_ <- size_ + 1
    For i <- size()-2 To pos Inc -1 Do
      set(i+1, get(i))
    set(pos, v)
    back_ <- cInc(back_, capacity())
  End-If
End.

Algorithm DArray::remove(pos:Integer) //O( n )
Begin
  For i <- pos TO size()-2 Inc 1 Do
    set(i, get(i + 1))
  back_ <-  cDec(back_, capacity())
  size_ <- size_ - 1
End.
```

Arrays

- Array dinámico circular.

```
Algorithm fold(out:Stream, data:CDArray[T]) //O(n )
Var
  i: Integer
Begin
   out.write(‘[‘)
   For i<-0 TO data.size()-1 Inc 1 Do
      out.write(‘ ‘, a.get(i))
   out.write(‘ ]’)
End.

Algorithm unfold(in:Stream):CDArray[T] //O(n )
Var
   token:String
   data:CDArray[t]
Begin
   in.read(token)
   If token = ’[‘ Then
      in.read(token)
      While Not in.eof() And token <> ’]’ Do
         data.pushBack(T(token))
         in.read(token)
      End-While
      If token <> ‘]’ Then
         Error (‘Wrong format’)
   Else
      Error (‘Worng format’)
   Return data
End.
```

"palabra", unidad mínima de significado

Resumiendo

- El Array está pensado principalmente para tener un acceso aleatorio con O(1) a costa de tener un tamaño fijo -> uso ineficiente de la memoria.
- El Array dinámico permite un uso más eficiente de la memoria al poder modificar su tamaño pero con coste O(N) y coste amortizado O(1).
- El Array dinámico circular permite crecer el array tanto por la cabeza como por la cola.

Referencias

- Lecturas recomendadas:
  - Caps. 8 y 9 de “Estructuras de Datos”, A. Carmona y otros. U. de Córdoba. 1999.
  - Caps 6 y 7 de *“Data structures and software develpment in an object oriented domain”*, Tremblay J.P. y Cheston, G.A. Prentice-Hall, 2001.
  - Wikipedia.