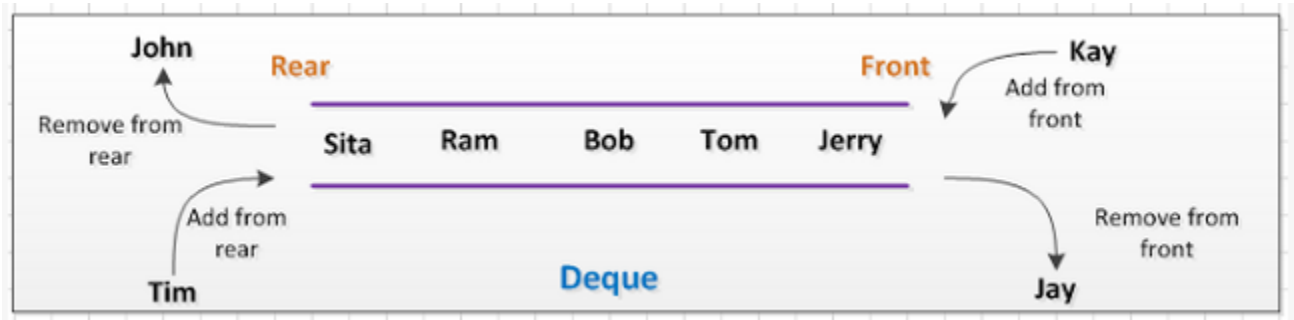
Teoria

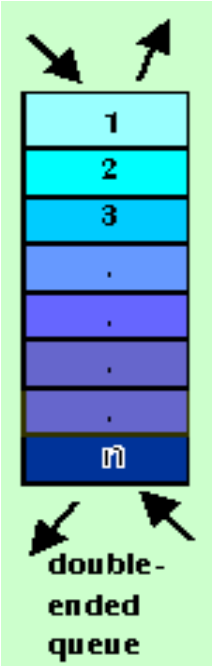
Double-Ended Queue

DEQue

DEQue

A Fila Circular com Dupla Terminação ou *DEQue* (*Double - Ended Queue*) é uma estrutura de dados em lista, à qual pode-se ter acesso pelas duas extremidades:





DEQue

Existem quatro funções principais que se aplicam em *DEQue*:

- *pushFront*: Insere um item à frente;

- *pushBack*: Insere um item atrás;

- *popFront*: Retira um item da frente; e

- *popBack*: Retira um item de trás.

DEQue

Entrada *DEQue* Saída

E D C B A

Início > < Fim

DEQue

Entrada *DEQue* Saída

F E D C B *pushBack* →

Início > A < Fim

_____

DEQue

Entrada	*DEQue*	Saída
F E D C B		

< Fim

A

Início >

_____

DEQue

Entrada *DEQue* Saída

G F E D C

pushBack →

B < Fim

A

Início >

DEQue

Entrada	*DEQue*	Saída
G F E D C		
	< Fim	
	B	
	A	
	Início >	
	_____	

DEQue

Entrada	*DEQue*	Saída
H G F E D	*pushFront* →	
	< Fim	
	B	
	A	
	Início > C	
	_____	

DEQue

Entrada	*DEQue*	Saída
H G F E D		
	< Fim	
	B	
	A	
	C	
	Início >	
	_____	

DEQue

Entrada	*DEQue*	Saída
H G F E	*pushBack* →	

D < Fim

B

A

C

Início >

_____

DEQue

Entrada	*DEQue*	Saída
H G F E		
	< Fim	
	D	
	B	
	A	
	C	
	Início >	
	_____	

DEQue

Entrada	*DEQue*	Saída
H G F		
pushFront →		
	< Fim	
	D	
	B	
	A	
	C	
	Início > E	

DEQue

Entrada	*DEQue*	Saída
H G F	< Fim	
	D	
	B	
	A	
	C	
	E	
	Início >	

DEQue

Entrada	*DEQue*	Saída
H G F	< Fim *popBack* →	D

B

A

C

E

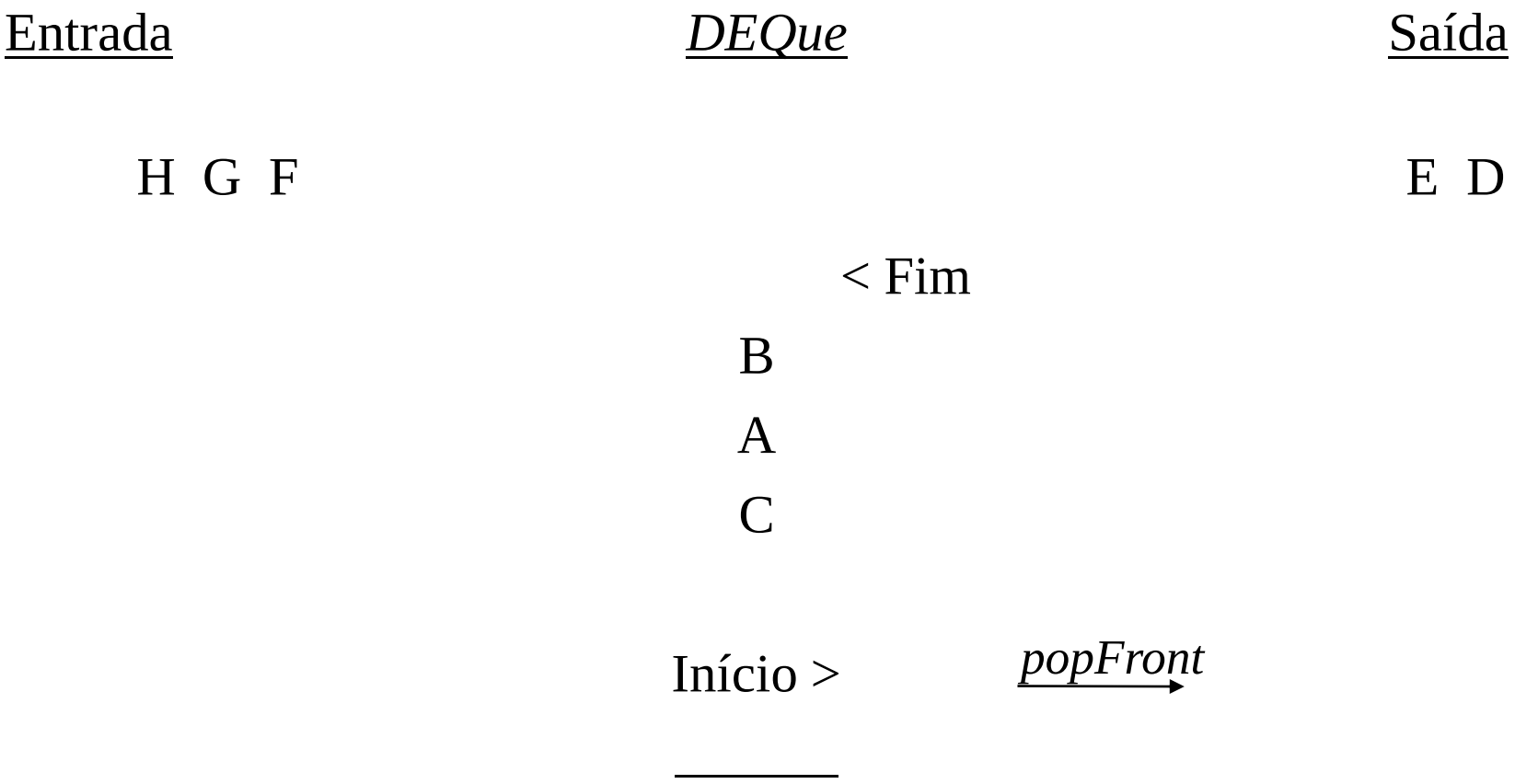
Início >

_____

DEQue

Entrada	*DEQue*	Saída
H G F	< Fim B A C E Início >	D

DEQue



DEQue

Entrada	*DEQue*	Saída
H G F		E D

< Fim

B

A

C

Início >

DEQue

Entrada | *DEQue* | Saída

H G F

B E D

< Fim *popBack* →

A

C

Início >

_____

DEQue

Entrada	*DEQue*	Saída
H G F		B E D

< Fim

A

C

Início >

DEQue

Entrada *DEQue* Saída

H G *pushBack* B E D

F < Fim

A

C

Início >

DEQue

Entrada	*DEQue*	Saída
H G		B E D

< Fim

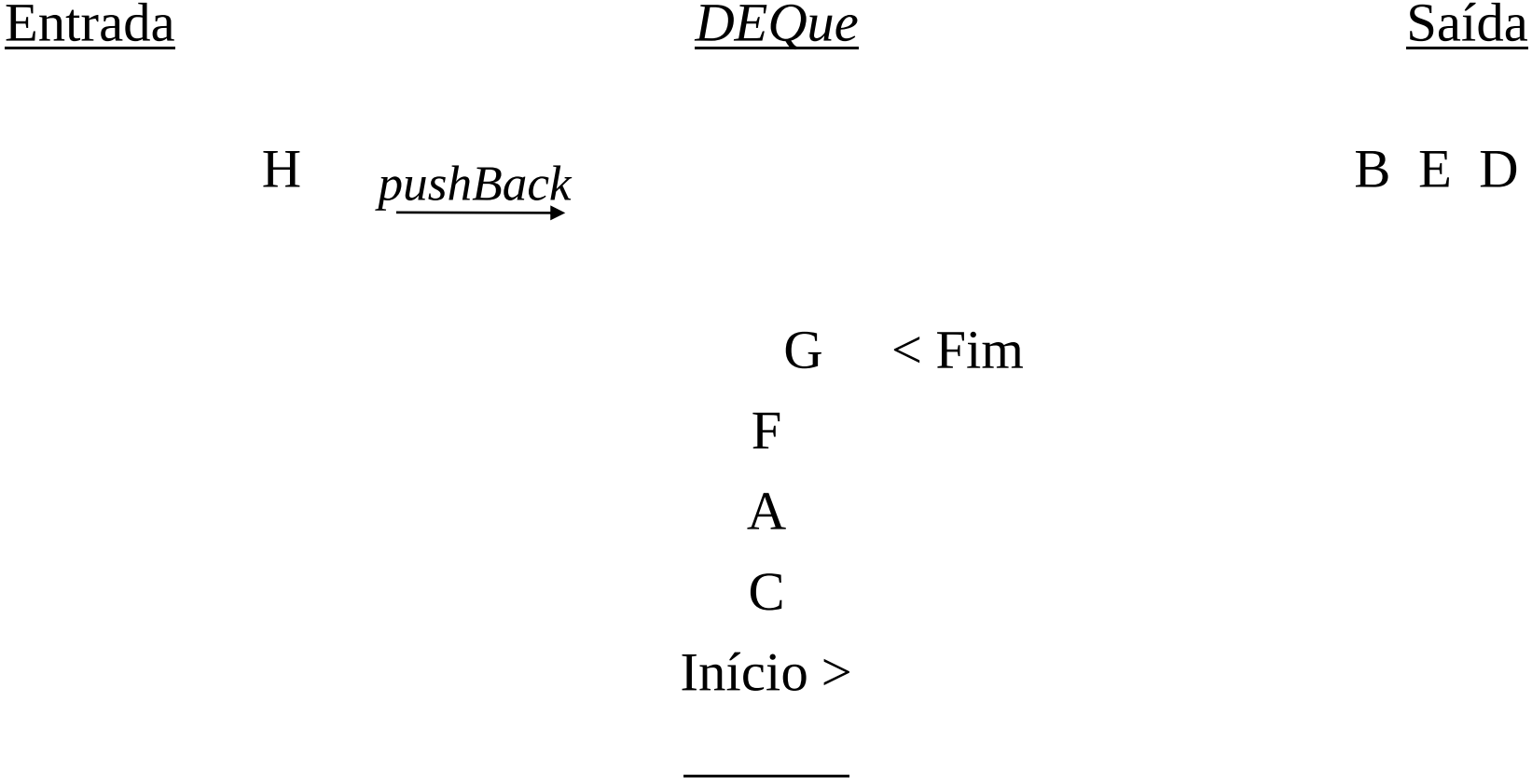
F

A

C

Início >

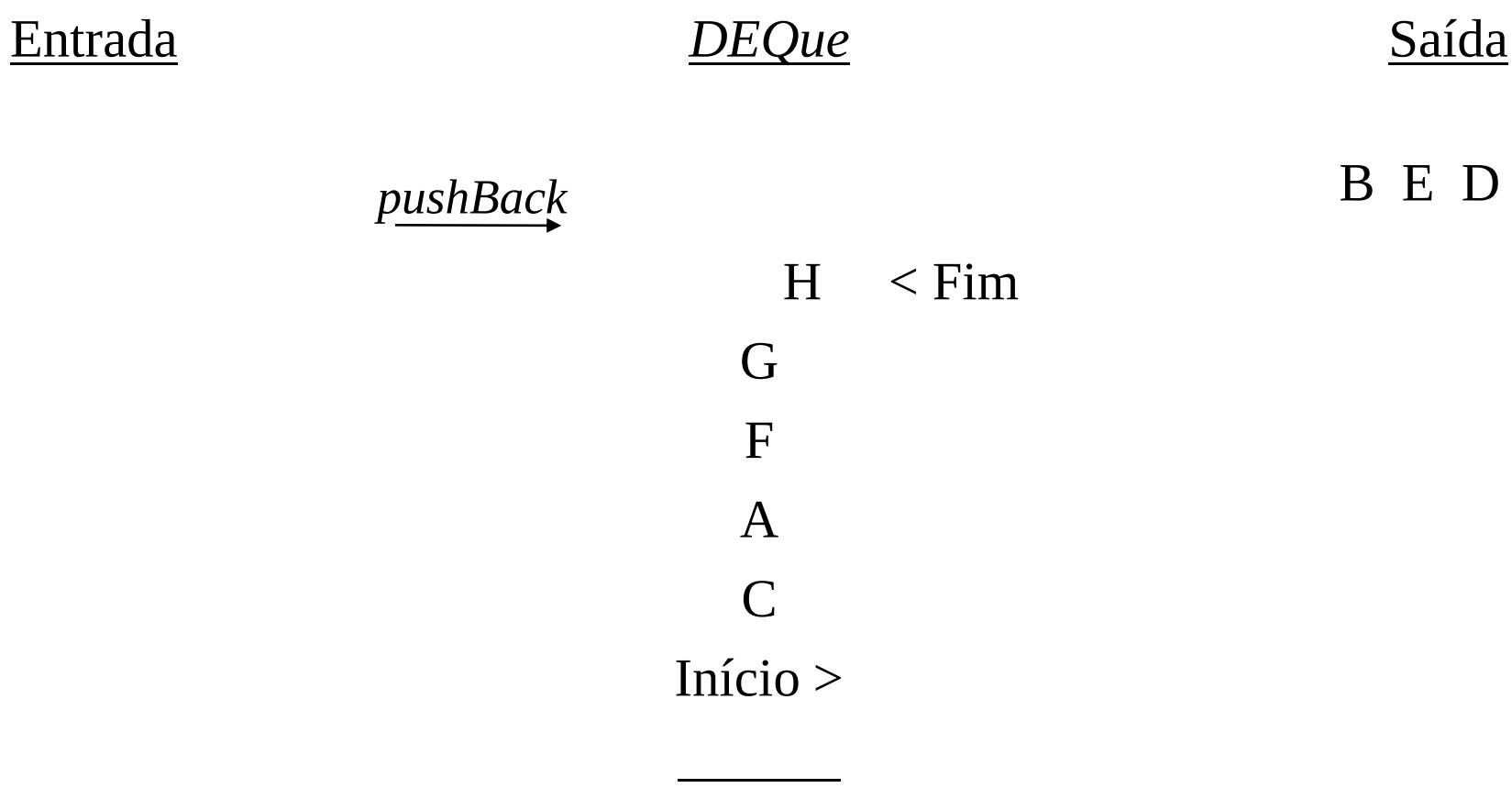
DEQue



DEQue

Entrada	*DEQue*	Saída
H	< Fim G F A C Início >	B E D

DEQue



DEQue

Entrada	*DEQue*	Saída
	< Fim	B E D
	H	
	G	
	F	
	A	
	C	
	Início >	

DEQue

Entrada	*DEQue*	Saída
	< Fim *popBack* →	H B E D
	G	
	F	
	A	
	C	
	Início >	

DEQue

Entrada	*DEQue*	Saída
		H B E D
	< Fim	
	G	
	F	
	A	
	C	
	Início >	
	_____	

DEQue

Entrada | *DEQue* | Saída

G H B E D

< Fim *popBack*

F

A

C

Início >

DEQue

Entrada	*DEQue*	Saída
		G H B E D

< Fim

F

A

C

Início >

_____

DEQue

Entrada | *DEQue* | Saída

C G H B E D

< Fim

F

A

Início >

popFront →

_____

DEQue

Entrada	*DEQue*	Saída
		C G H B E D

< Fim

F

A

Início >

_____

DEQue

Entrada *DEQue* Saída

A C G H B E D

< Fim

F

Início >

popFront →

_____

DEQue

Entrada *DEQue* Saída

A C G H B E D

< Fim

F

Início >

DEQue

Entrada *DEQue* Saída

F A C G H B E D

< Fim

Início > *popFront*

_____

DEQue

Entrada *DEQue* Saída

F A C G H B E D

Início > < Fim

DEQue

Entrada *DEQue* Saída

H G F E D C B A

alteração da ordem

para qualquer quantidade

F A C G H B E D

DEQue

Exemplos de utilização de *DEQue*:

- Execução de processos prioritários em Sistemas Distribuídos;
- Alteração da ordem de informações armazenadas; e
- Retirada de um elemento específico de uma estrutura de dados.

DEQue

Outras funções que se aplicam a *DEQue*:

- *size* : Informa o tamanho;

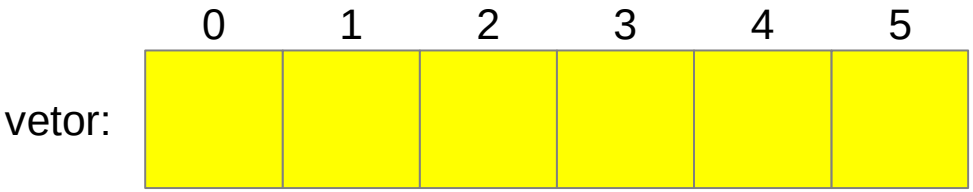
- *front*: Informa o elemento do início, sem retirá-lo;

- *back*: Informa o elemento do fim, sem retirá-lo;

- *isEmpty*: Informa se a *DEQue* está vazia; e

- *isFull*: Informa se a *DEQue* está cheia.

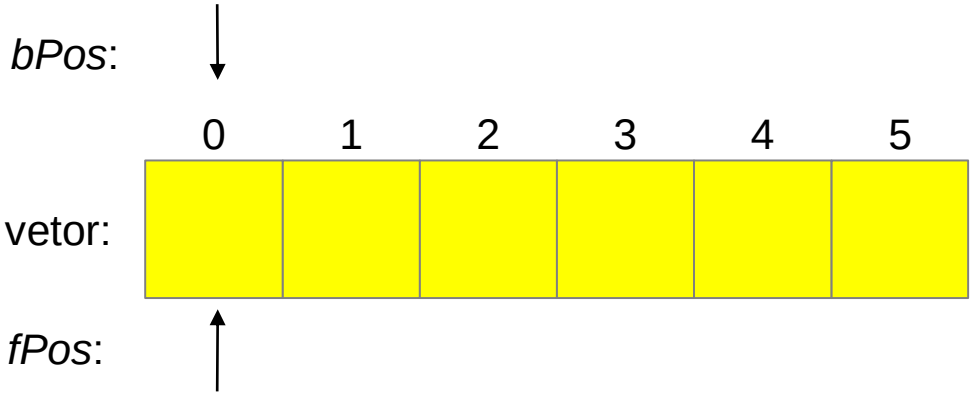
DEQue



int vetor[ ] = new vetor[6];

Vetor com o tamanho máximo necessário

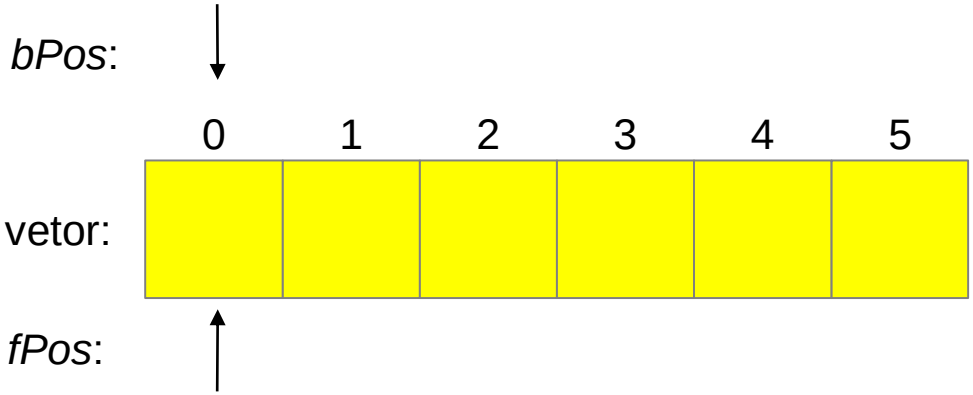
DEQue




```
int bPos, fPos;
:
:
bPos = fPos = 0;
```

Inicializa indicadores de posição (*front* e *back*)

DEQue

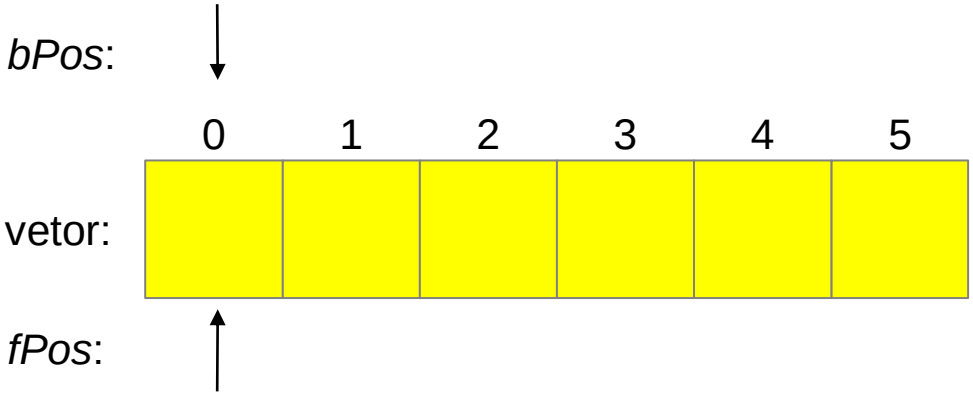


bDEQueFull: false

```
boolean bDEQueFull;
:
:
bDEQueFull = false;
```

Inicializa indicador de *DEQue* cheia

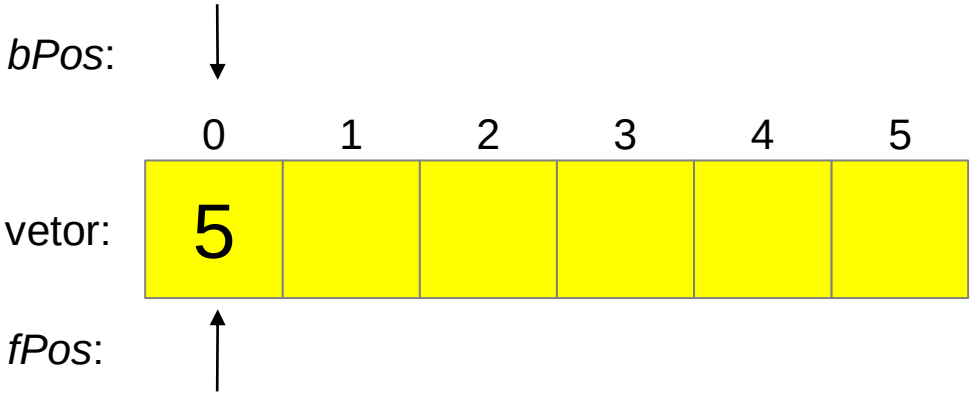
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(5);
```

Verifica se está cheia

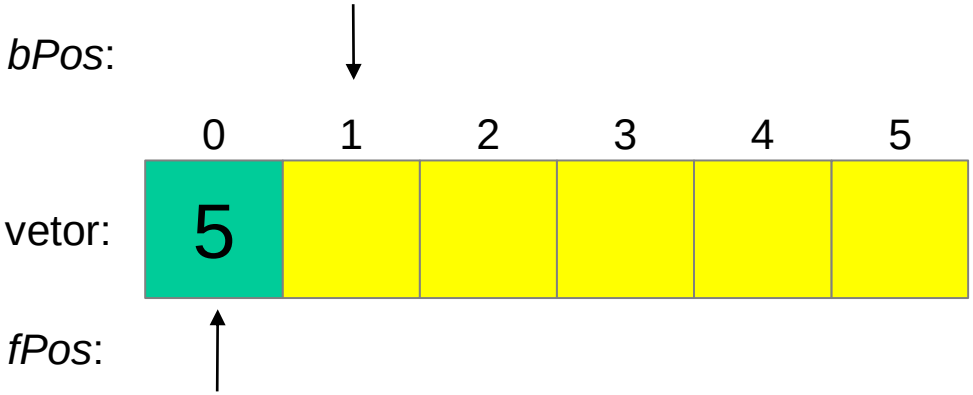
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(5);
```

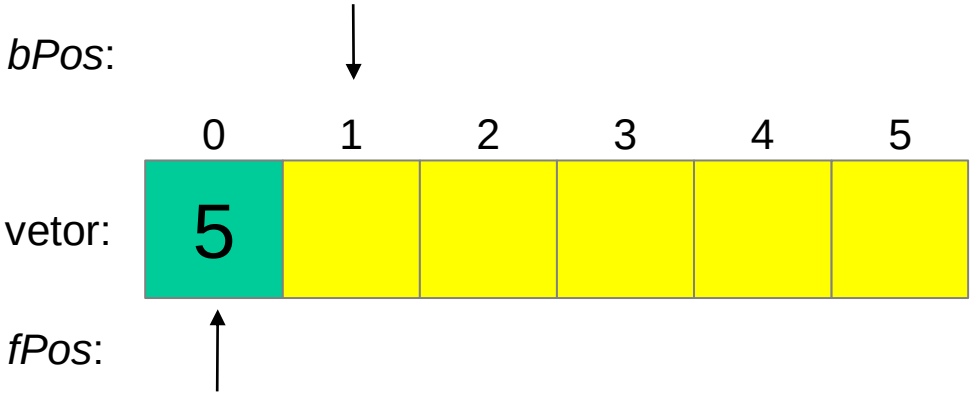
Não está cheia! Coloca o elemento 5

DEQue



Incrementa bPos (*back*)

DEQue



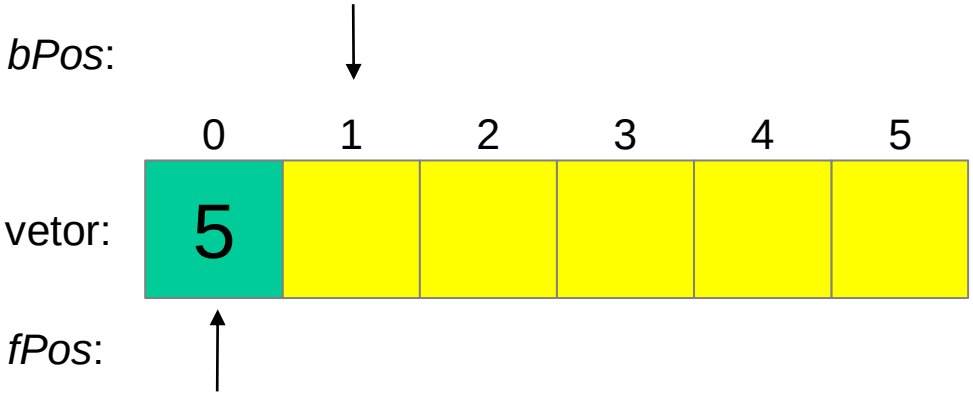
bDEQueFull: false

```
int frente, atras;
:
frente = front( );
atras = back( );

// frente: 5
// atras: 5
```

Consulta a frente

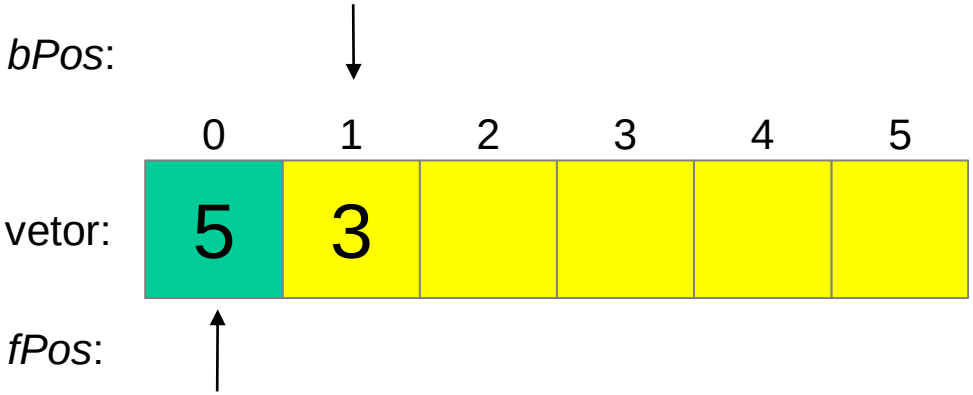
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(3);
```

Verifica se está cheia

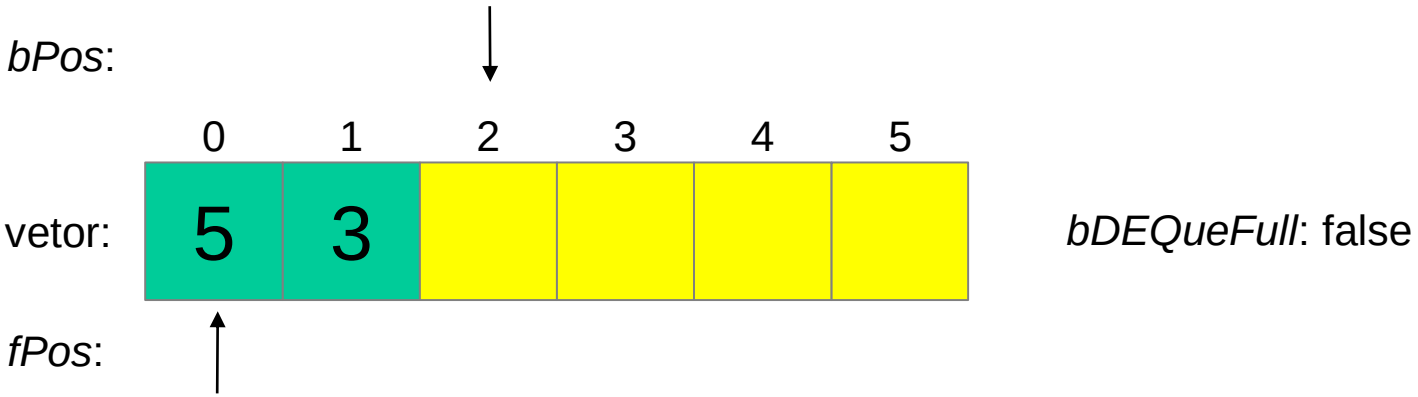
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(3);
```

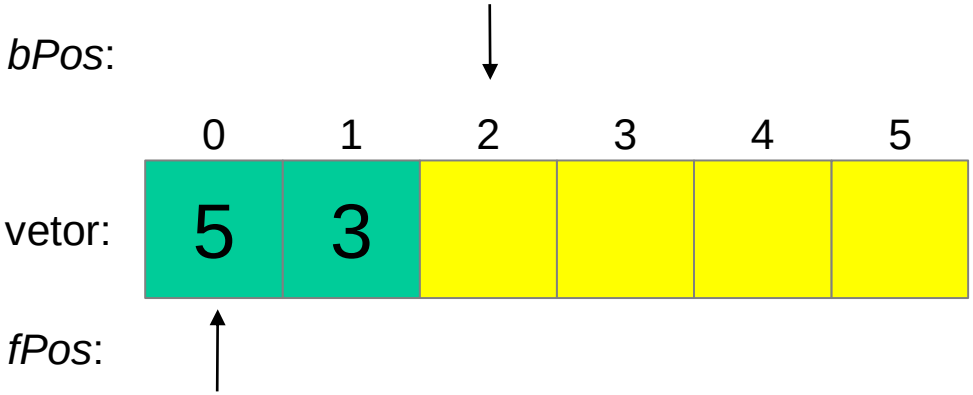
Não está cheia! Coloca o elemento 3

DEQue



Incrementa bPos (*back*)

DEQue



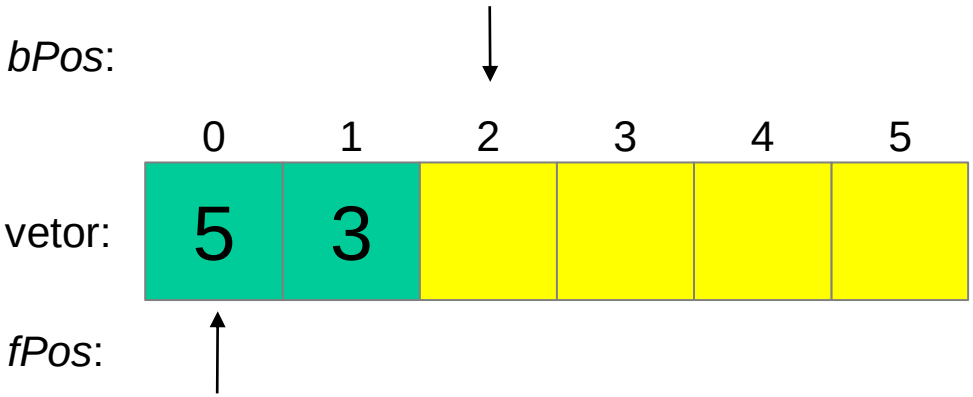
bDEQueFull: false

```
int frente, atras;
:
frente = front( );
atras = back( );

// frente: 5
// atras: 3
```

Consulta a frente

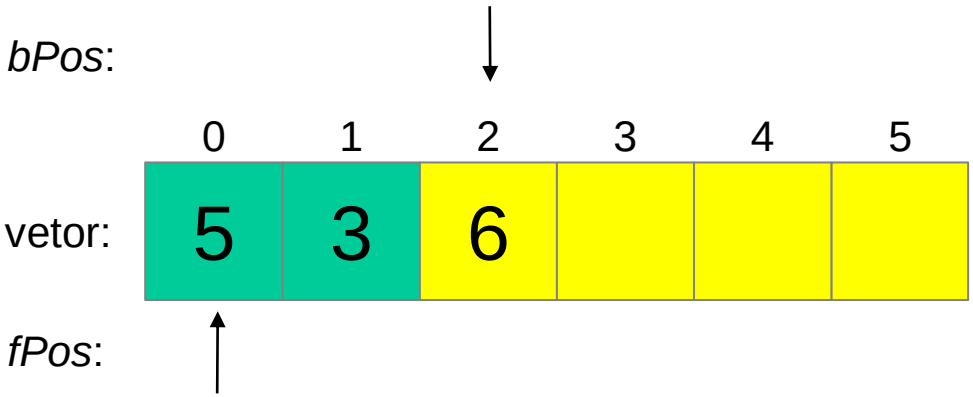
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(6);
```

Verifica se está cheia

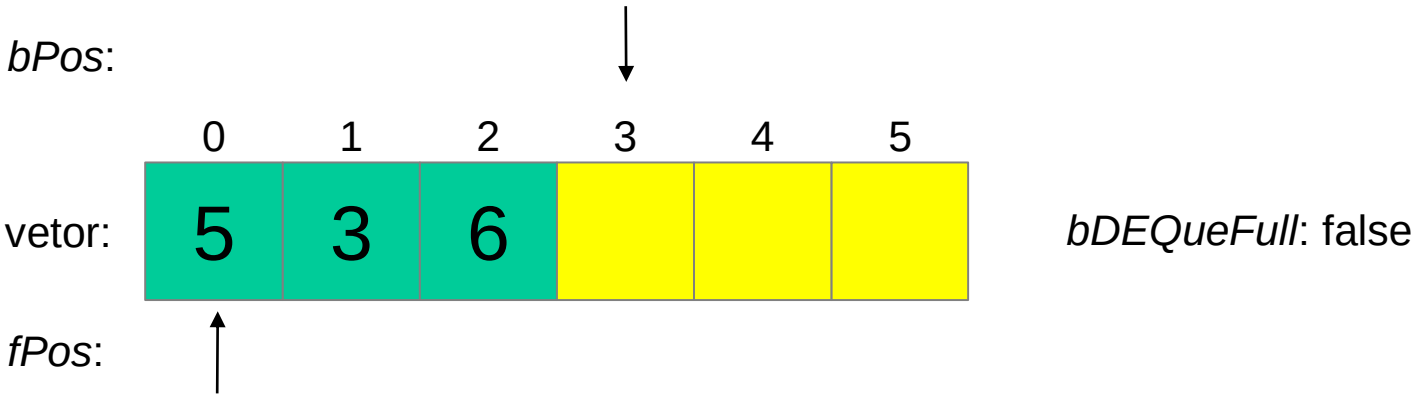
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(6);
```

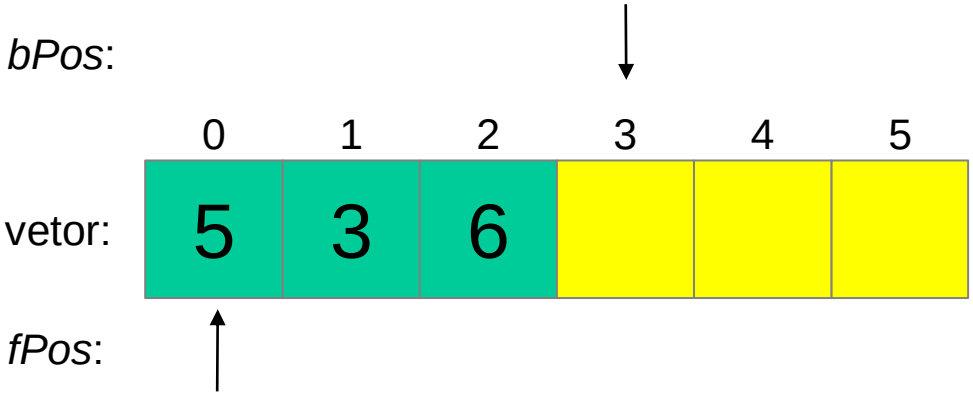
Não está cheia! Coloca o elemento 6

DEQue



Incrementa bPos (*back*)

DEQue



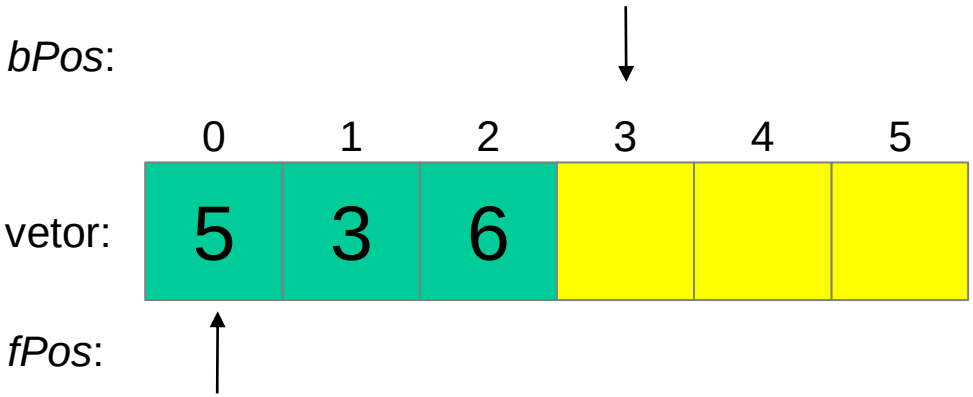
bDEQueFull: false

```
int frente, atras;
:
frente = front( );
atras = back( );

// frente: 5
// atras: 6
```

Consulta a frente

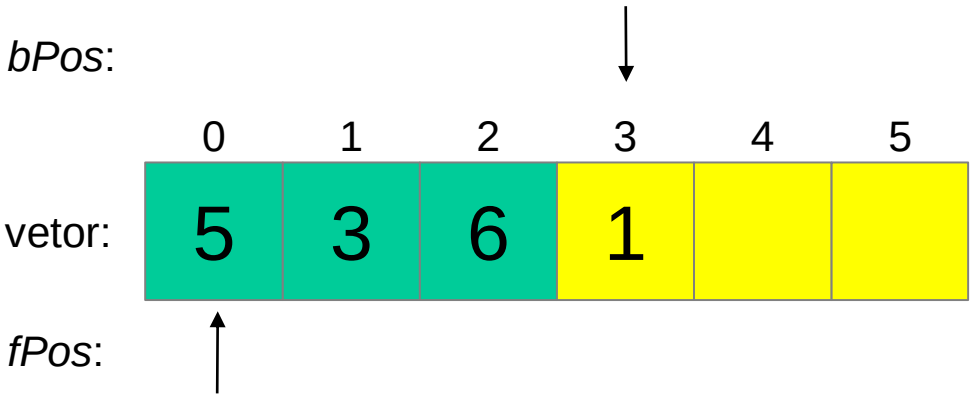
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(1);
```

Verifica se está cheia

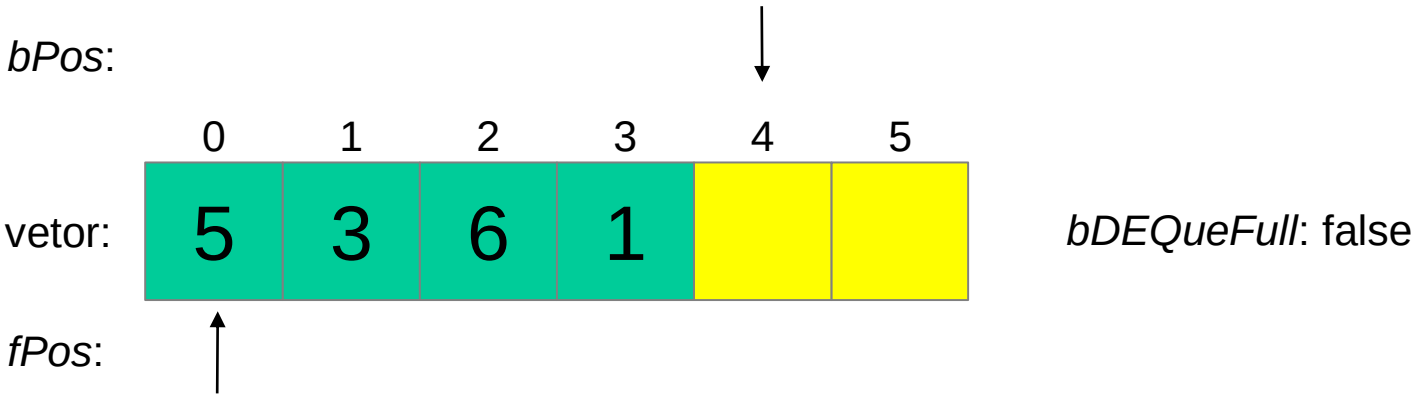
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(1);
```

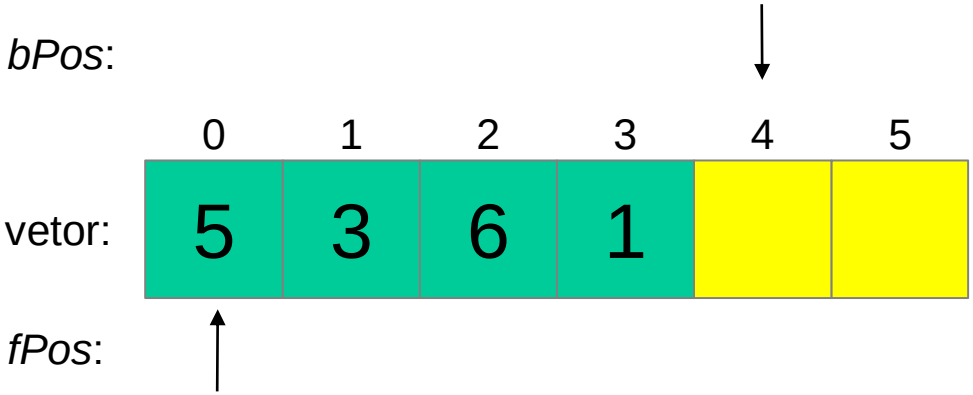
Não está cheia! Coloca o elemento 1

DEQue



Incrementa bPos (*back*)

DEQue



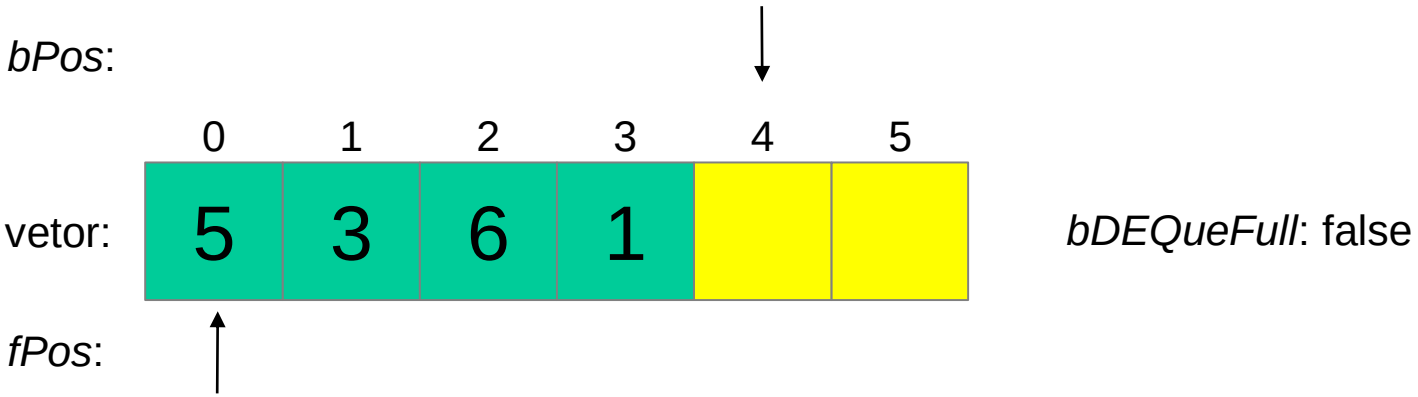
bDEQueFull: false

```
int frente, atras;
:
frente = front( );
atras = back( );

// frente: 5
// atras: 1
```

Consulta a frente

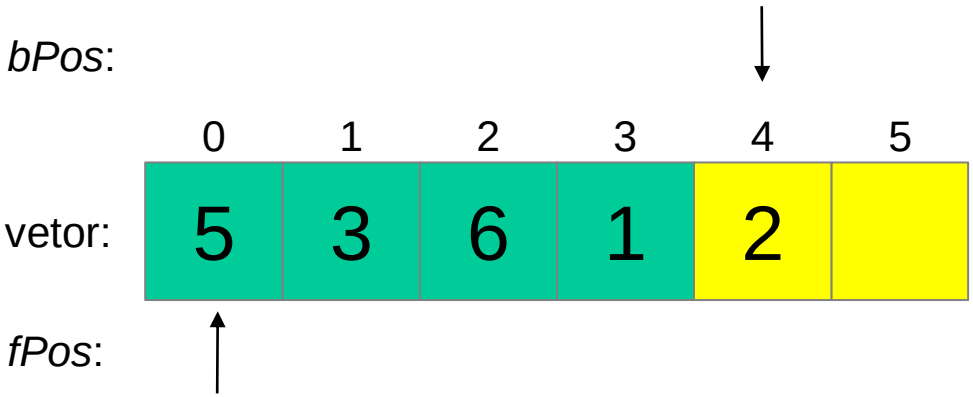
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(2);
```

Verifica se está cheia

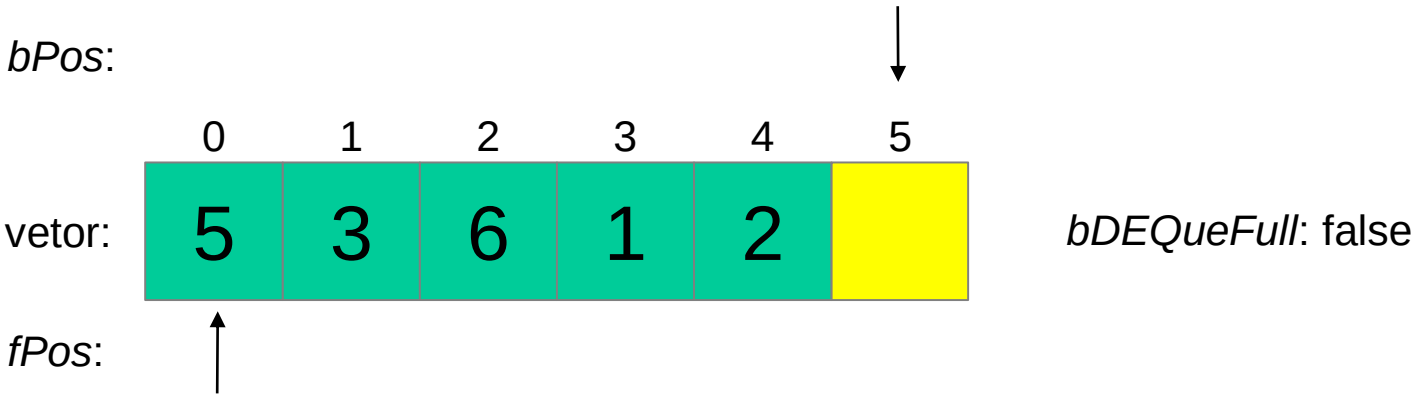
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(2);
```

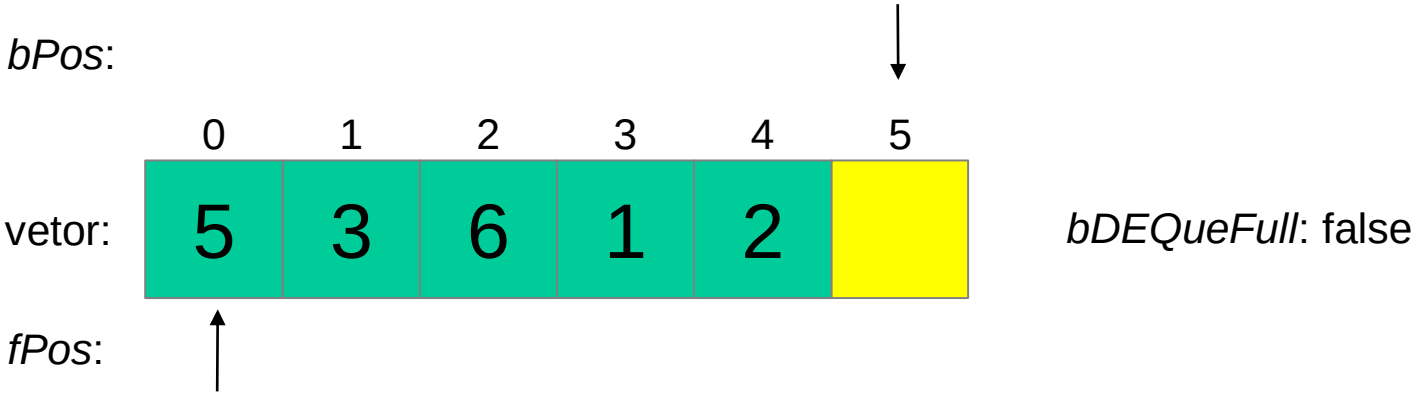
Não está cheia! Coloca o elemento 2

DEQue



Incrementa bPos (*back*)

DEQue

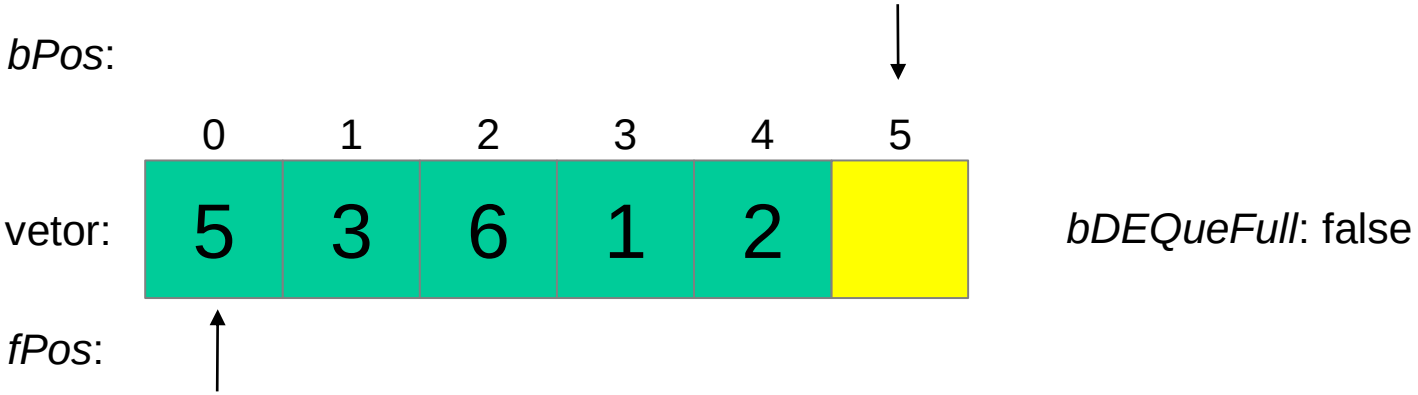



```
int frente, atras;
:
frente = front( );
atras = back( );

// frente: 5
// atras: 2
```

Consulta a frente

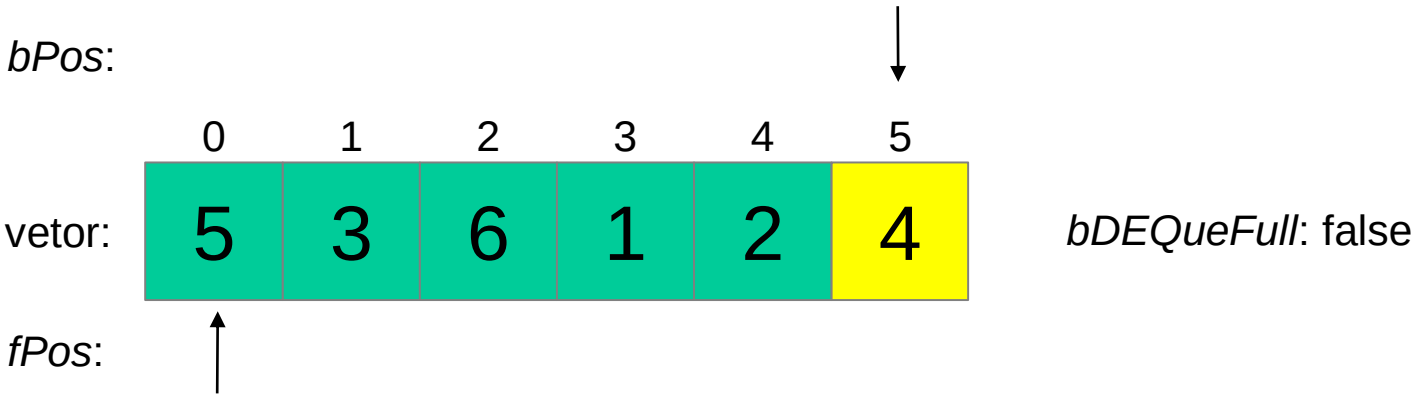
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(4);
```

Verifica se está cheia

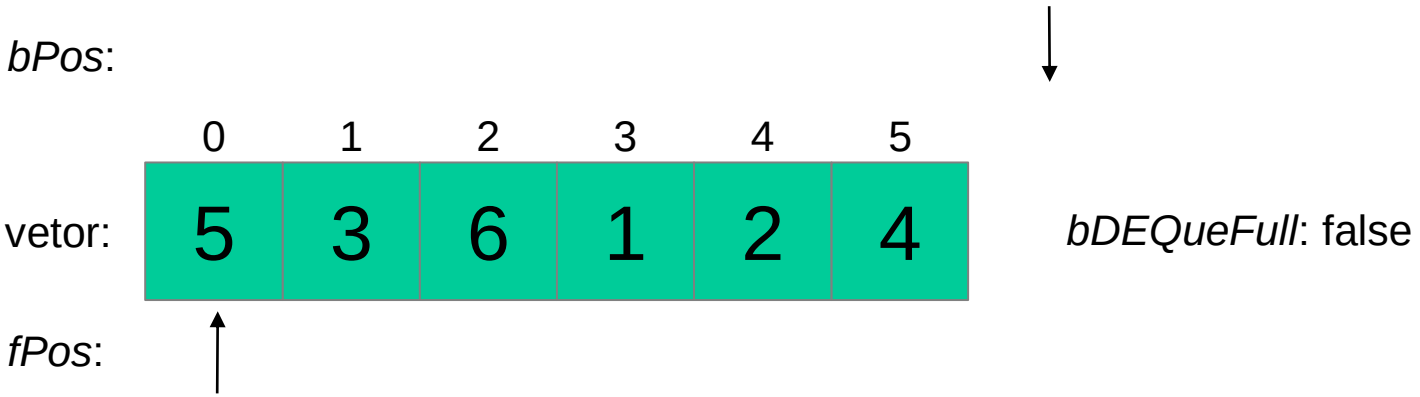
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(4);
```

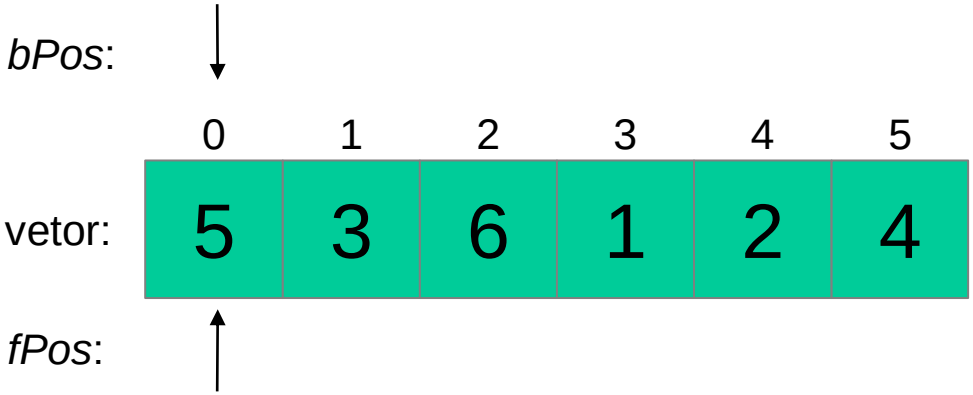
Não está cheia! Coloca o elemento 4

DEQue



Incrementa bPos (*back*)

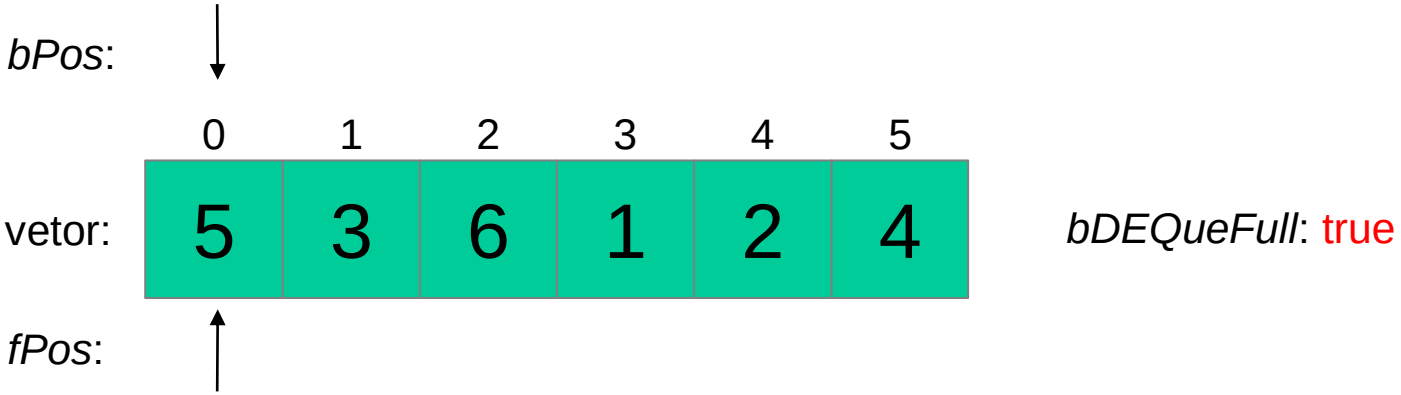
DEQue




```
if ( bPos >= vetor.length )
    bPos = 0;
else
    bPos++;
```

Ajusta bPos (*back*)

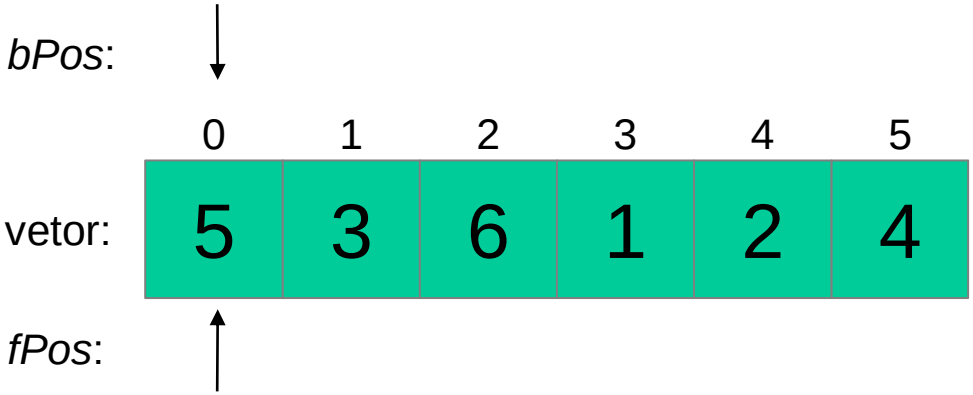
DEQue




```
if ( bPos == fPos )
    bDEQueFull = true;
```

Indica que a *DEQue* está cheia

DEQue

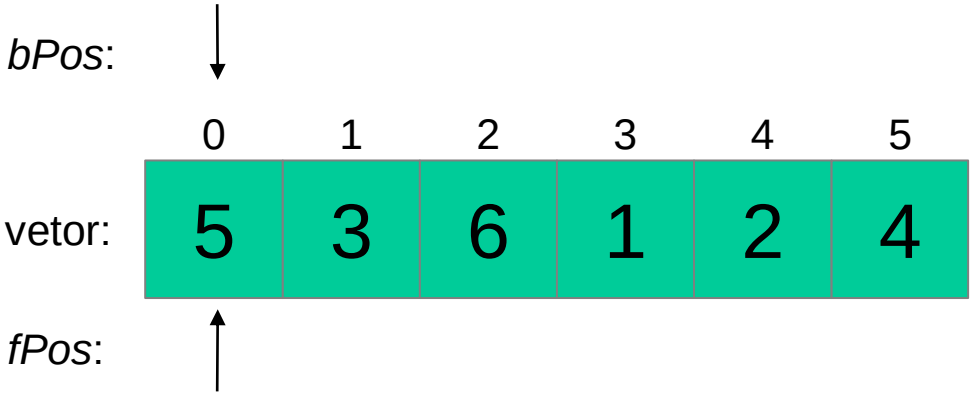



```
int frente, atras;
:
frente = front( );
atras = back( );

// frente: 5
// atras: 4
```

Consulta a frente

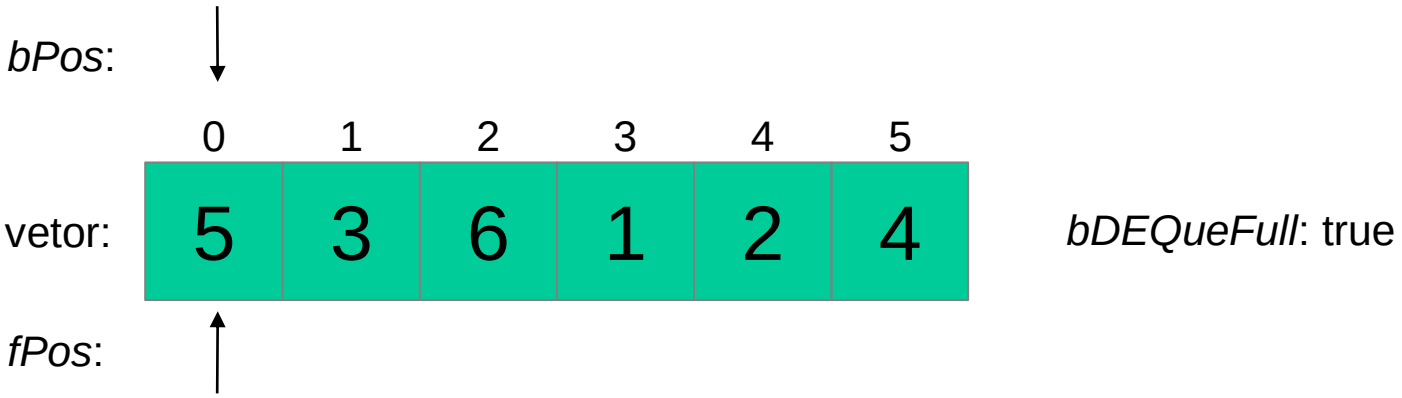
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(7);
```

Verifica se está cheia

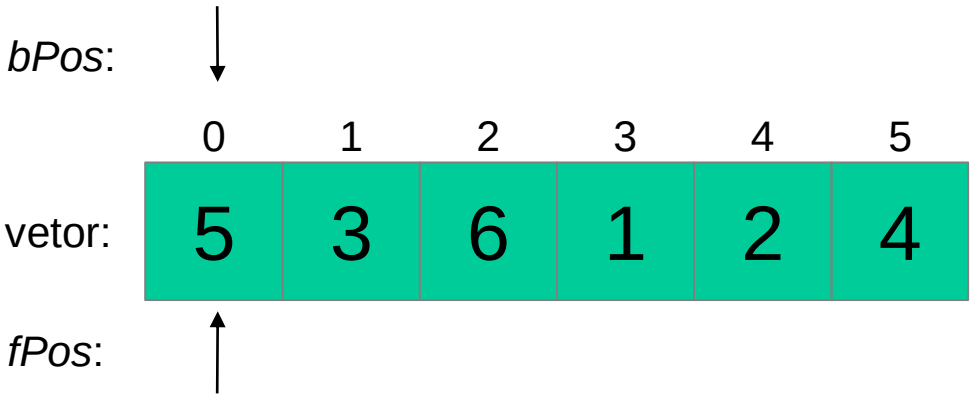
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(7);
```

Está cheia! Não coloca o elemento 7

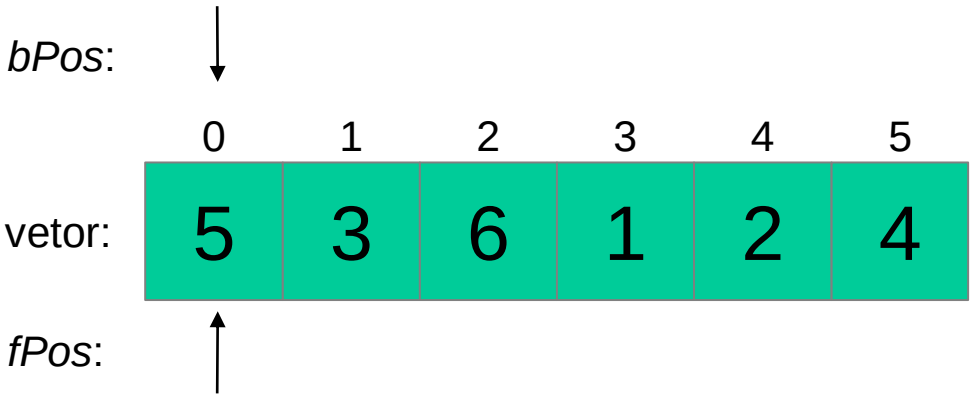
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popFront( );
```

Verifica se está vazia

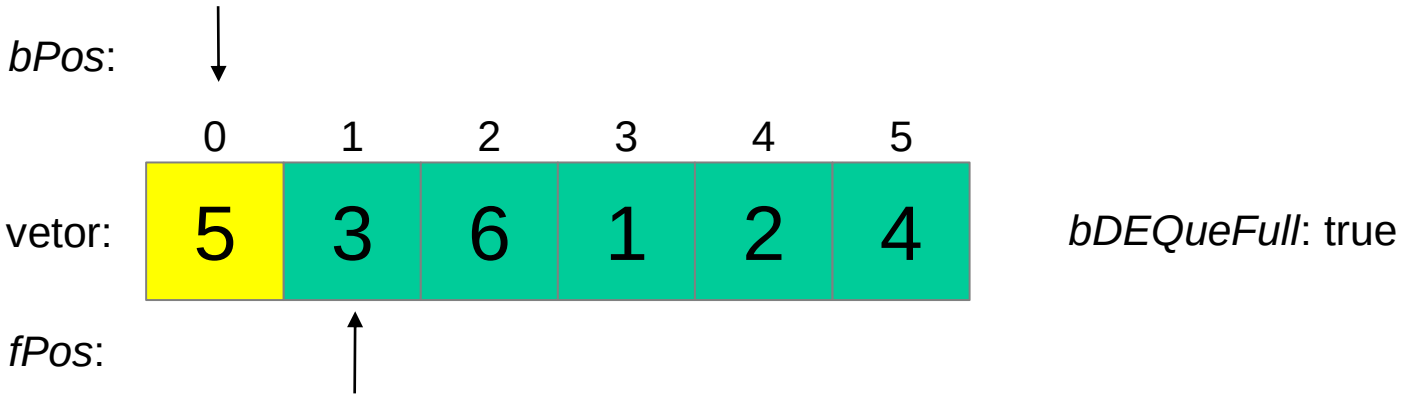
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popFront( );
// n: 5
```

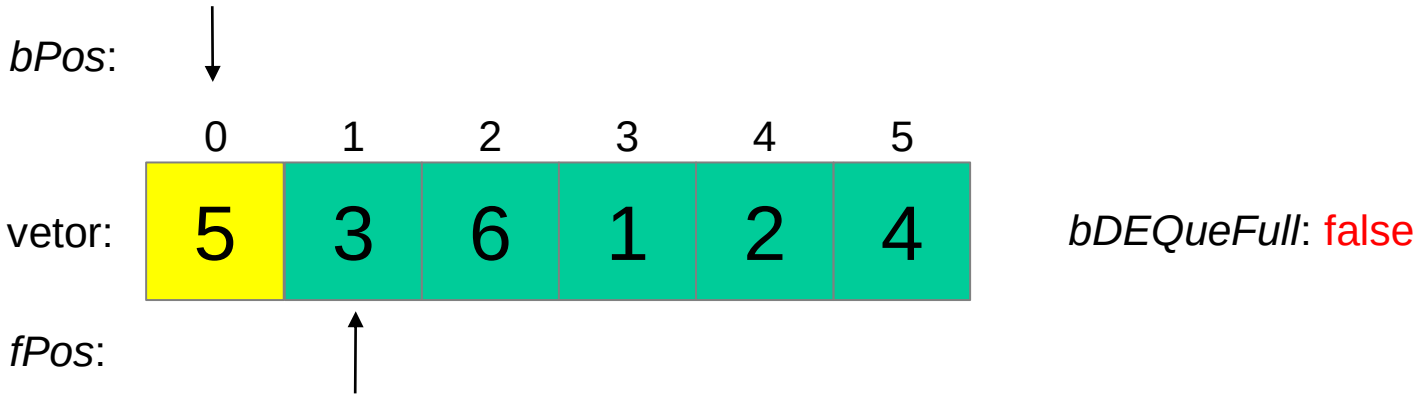
Não está vazia! Retira o elemento da frente

DEQue



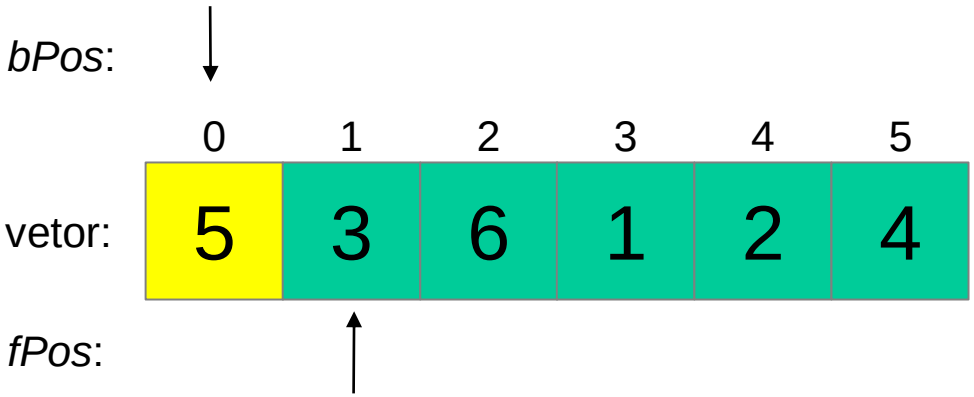
Incrementa fPos (*front*)

DEQue



Indica que a *DEQue* não está cheia

DEQue



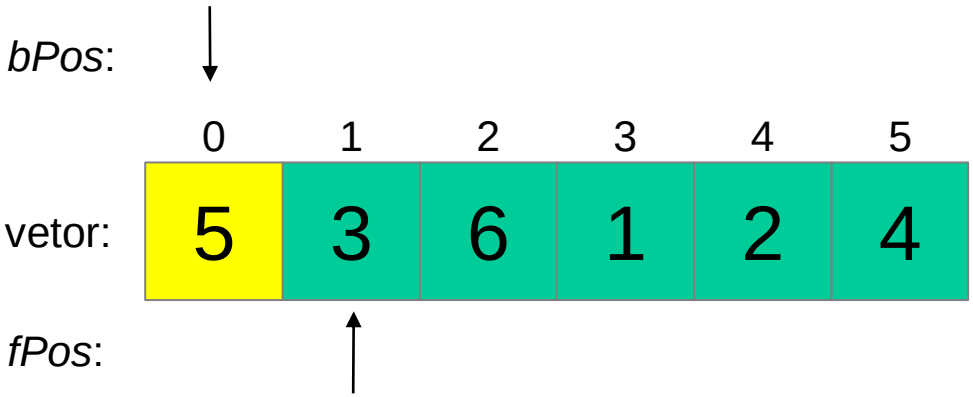
bDEQueFull: false

```
int frente, atras;
:
frente = front( );
atras = back( );

// frente: 3
// atras: 4
```

Consulta a frente

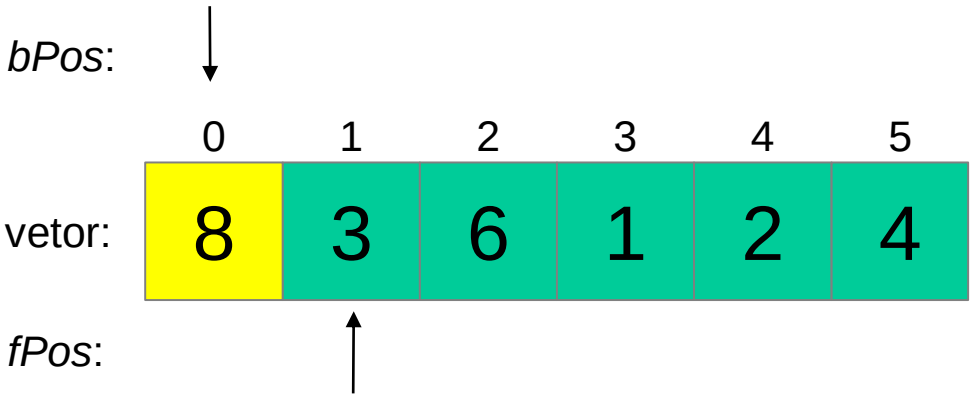
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(8);
```

Verifica se está cheia

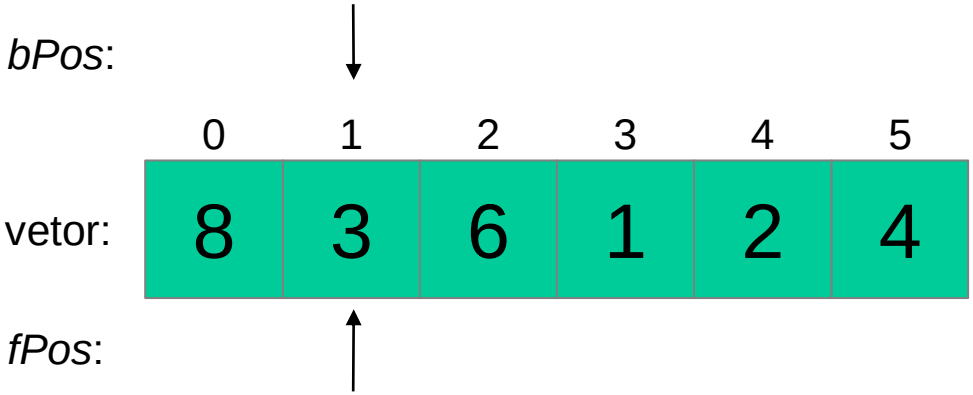
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushBack(8);
```

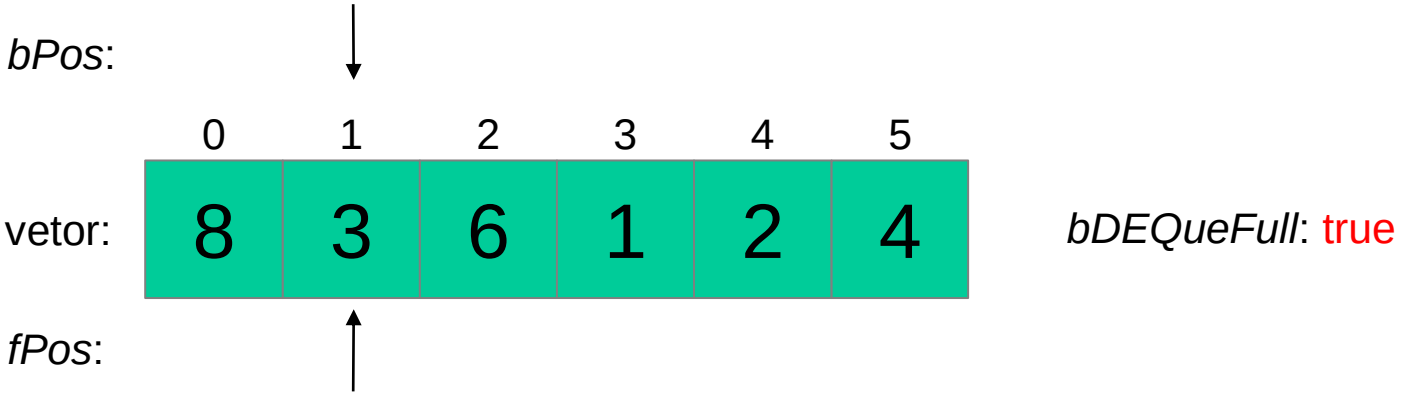
Não está cheia! Coloca o elemento 8

DEQue



Incrementa bPos (*back*)

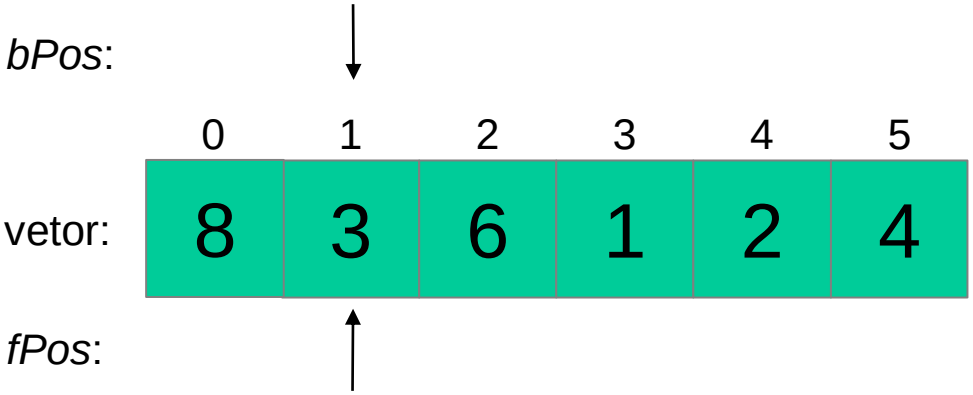
DEQue




```
if ( bPos == fPos )
    bDEQueFull = true;
```

Indica que a *DEQue* está cheia

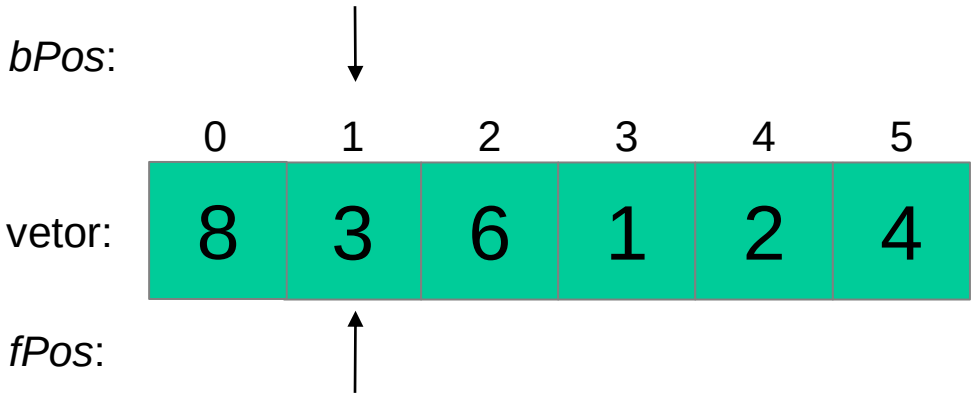
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popFront( );
```

Verifica se está vazia

DEQue

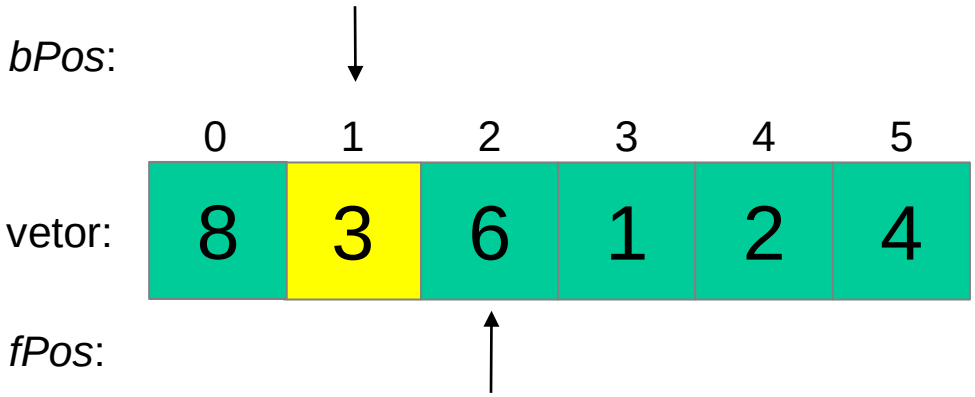


bDEQueFull: false

```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popFront( );
// n: 3
```

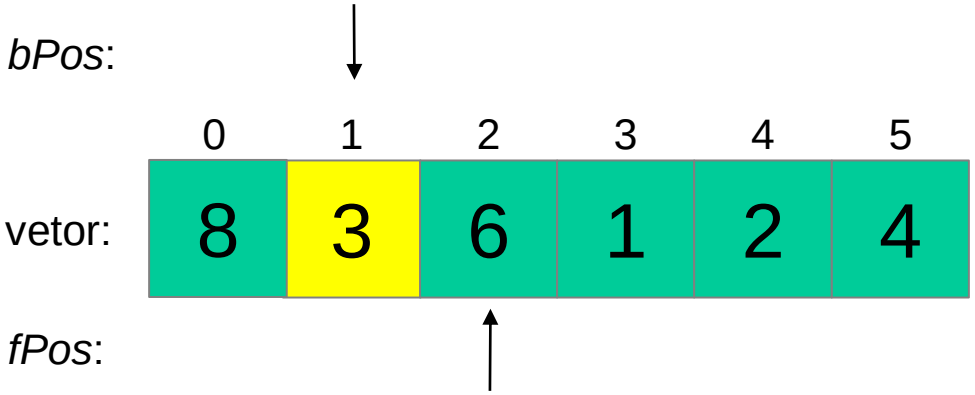
Não está vazia! Retira o elemento da frente

DEQue



Incrementa fPos (*front*)

DEQue



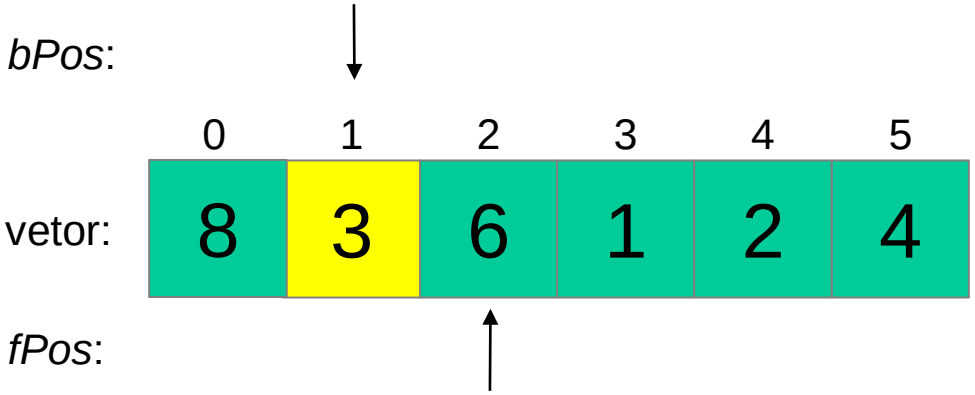
bDEQueFull: false

```
int frente, atras;
:
frente = front( );
atras = back( );

// frente: 6
// atras: 8
```

Consulta a frente

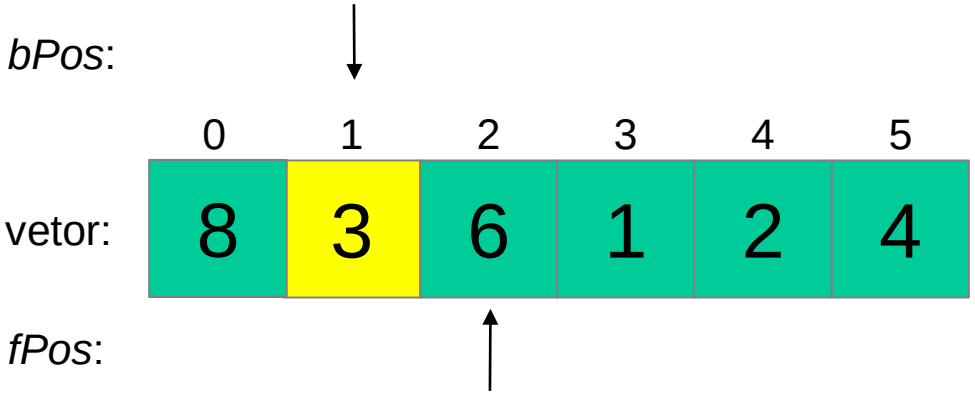
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popFront( );
```

Verifica se está vazia

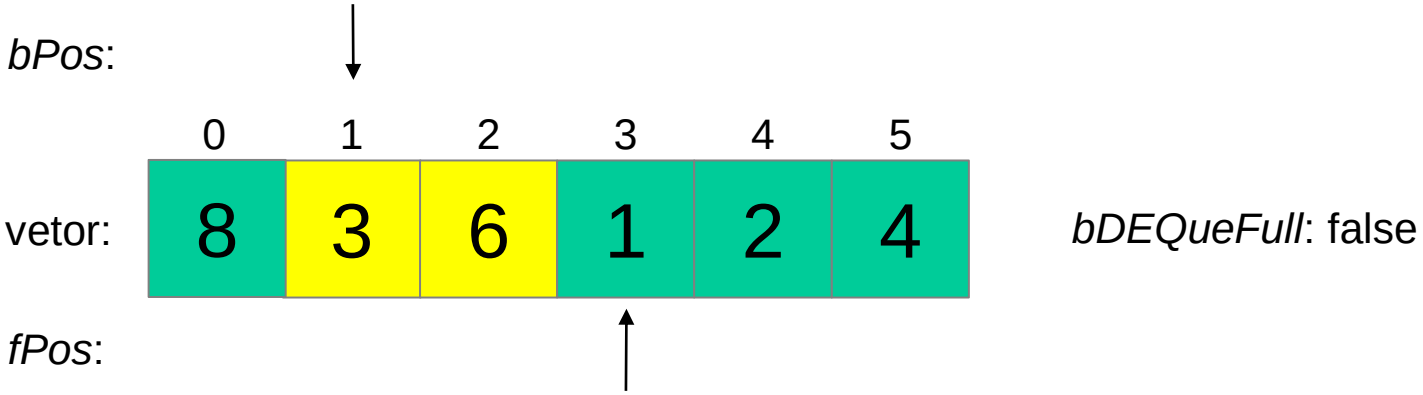
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popFront( );
// n: 6
```

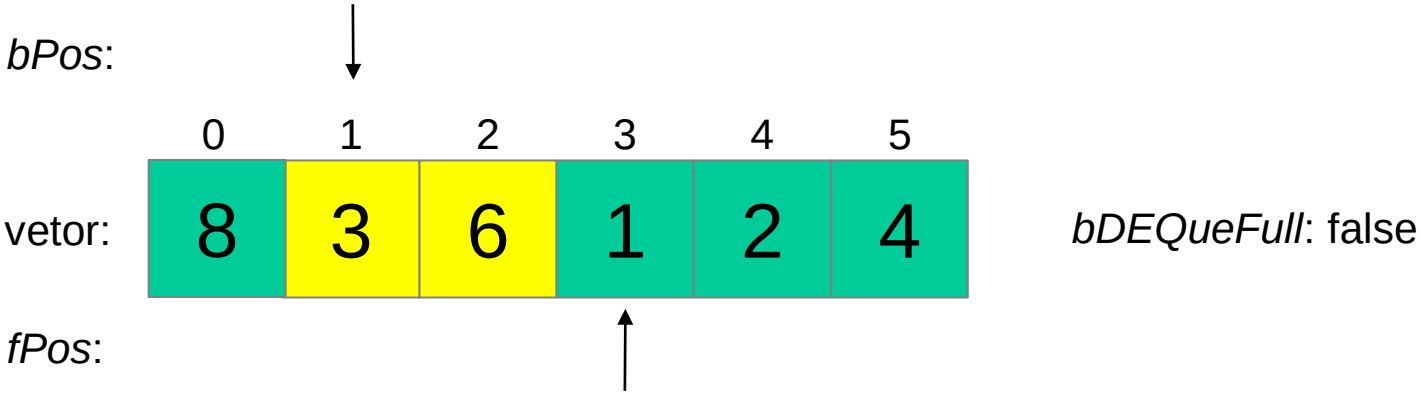
Não está vazia! Retira o elemento da frente

DEQue



Incrementa fPos (*front*)

DEQue

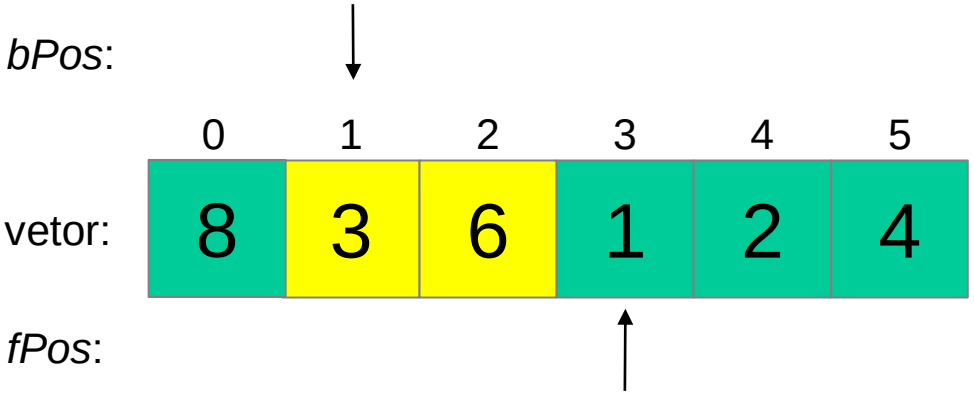



```
int frente, atras;
:
frente = front( );
atras = back( );

// frente: 1
// atras: 8
```

Consulta a frente

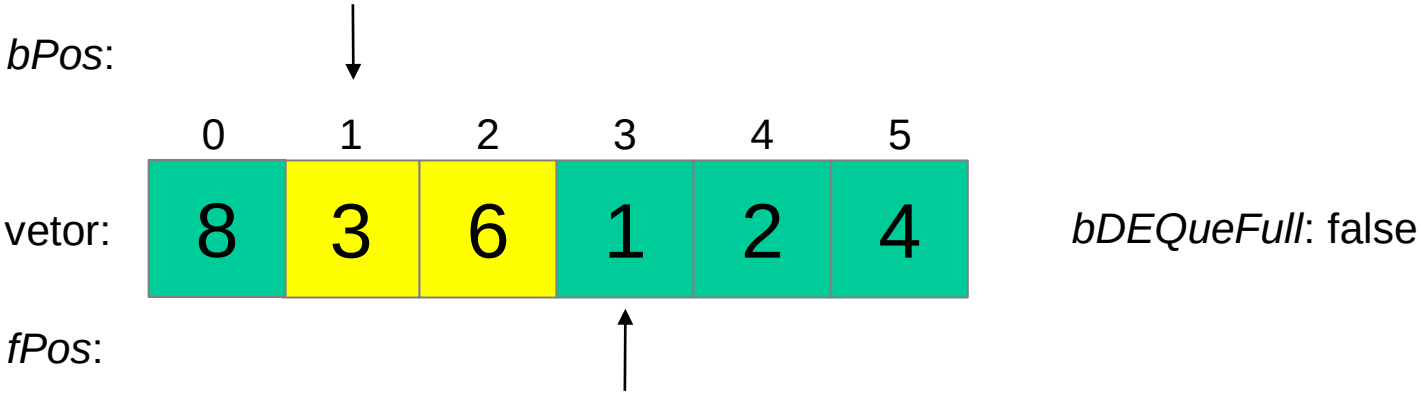
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popFront( );
```

Verifica se está vazia

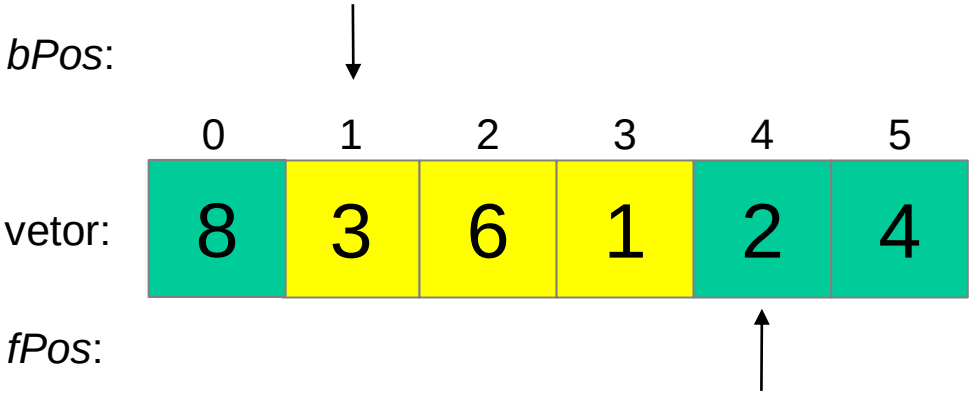
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popFront( );
// n: 1
```

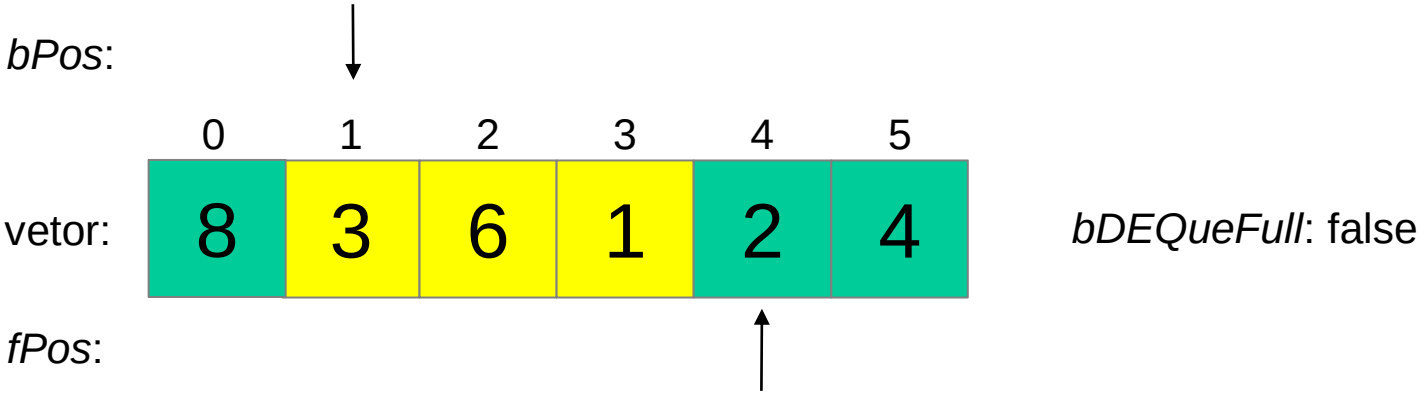
Não está vazia! Retira o elemento da frente

DEQue



Incrementa fPos (*front*)

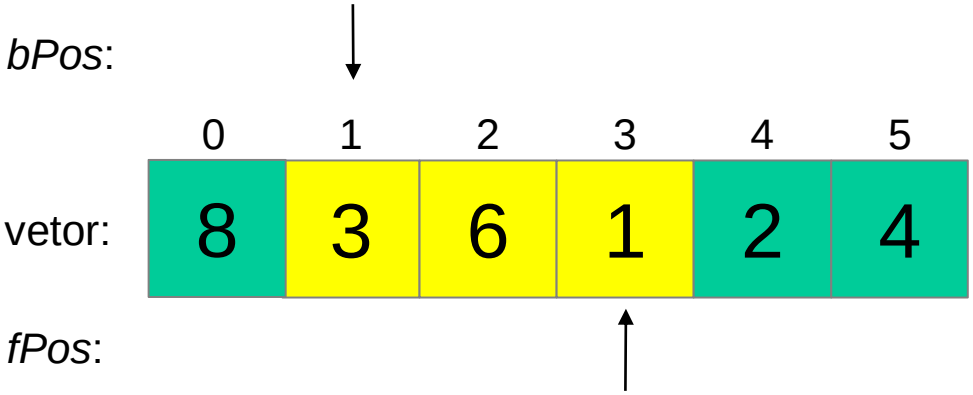
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushFront(5);
```

Verifica se está cheia

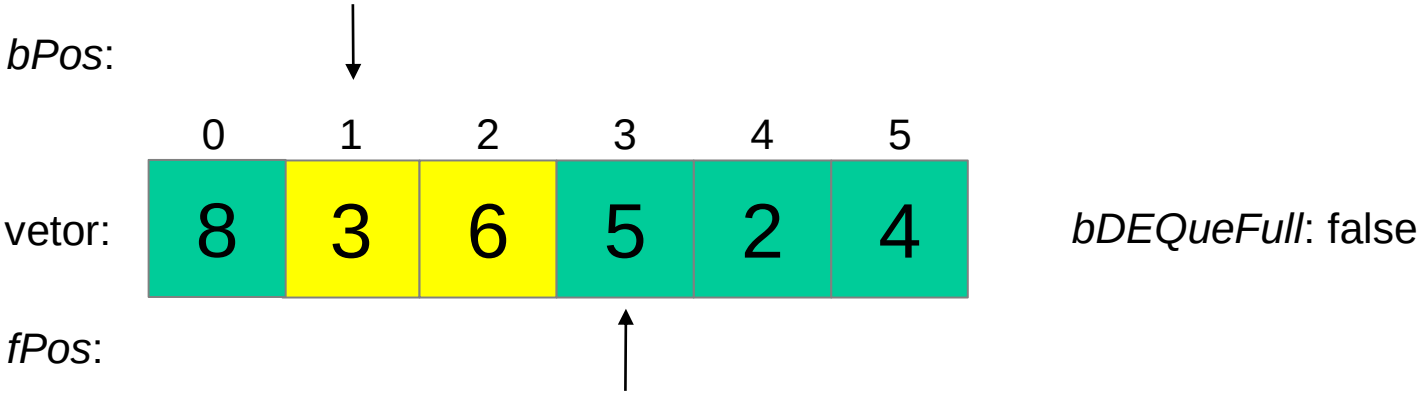
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushFront(5);
```

Não está cheia! Decrementa fPos (*front*)

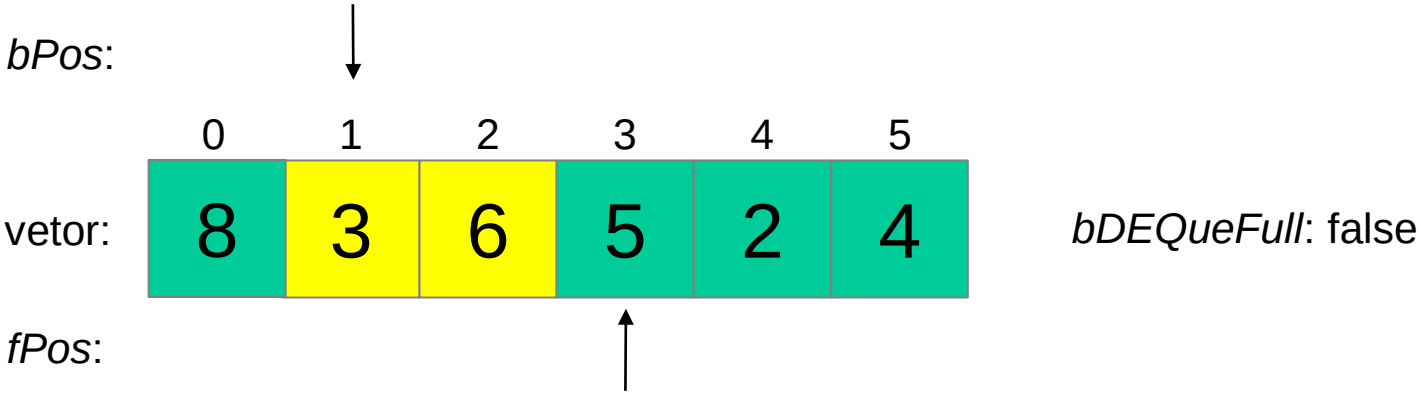
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushFront(5);
```

Coloca o elemento 5

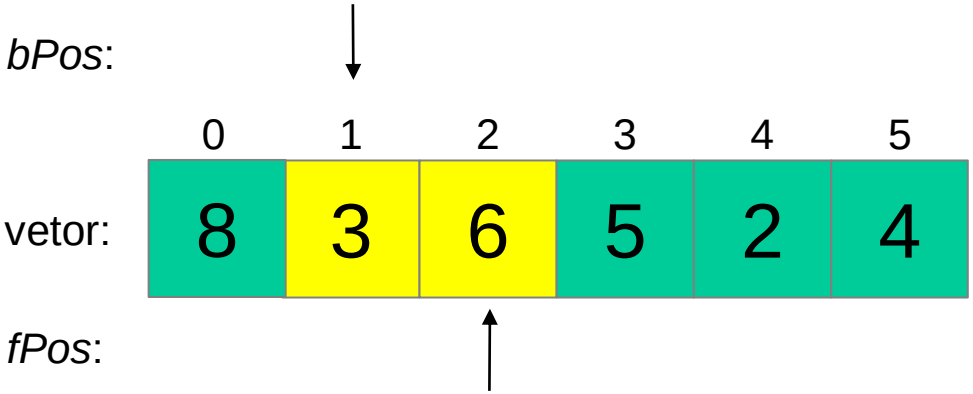
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushFront(9);
```

Verifica se está cheia

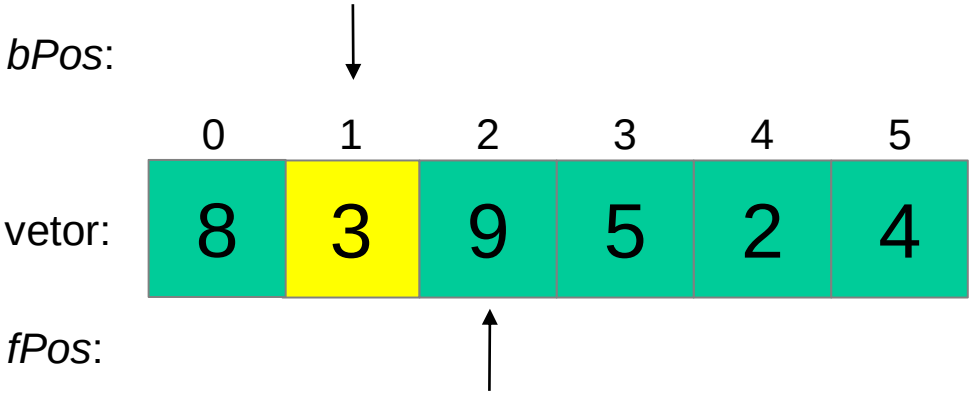
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushFront(9);
```

Não está cheia! Decrementa fPos (*front*)

DEQue

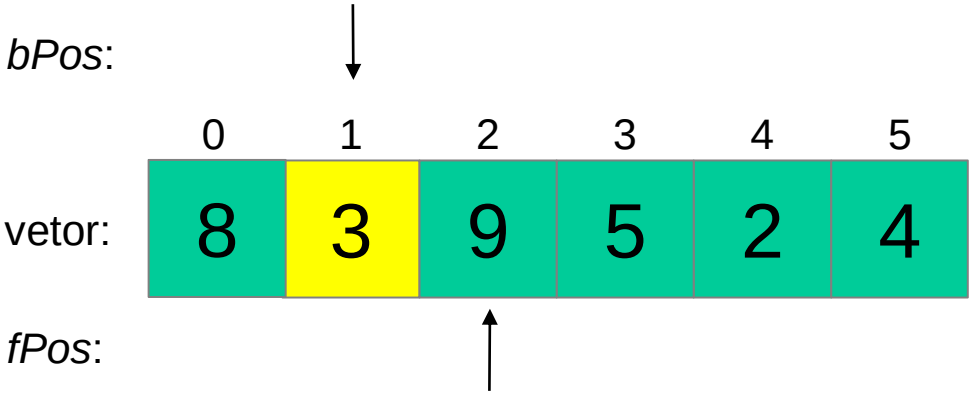


bDEQueFull: false

```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushFront(9);
```

Coloca o elemento 9

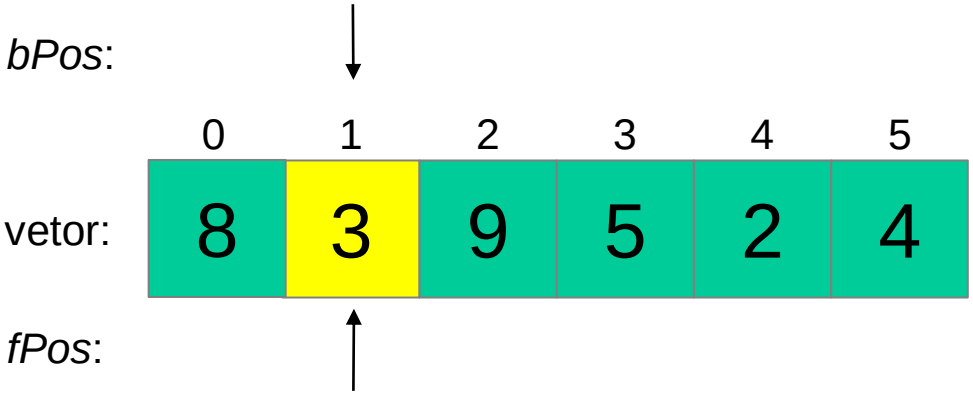
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushFront(1);
```

Verifica se está cheia

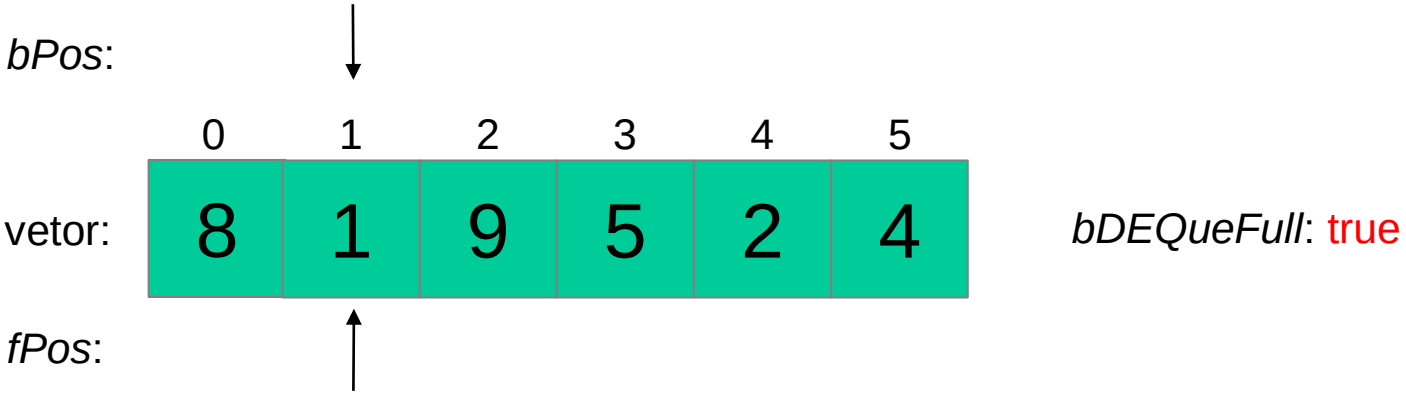
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushFront(1);
```

Não está cheia! Decrementa fPos (*front*)

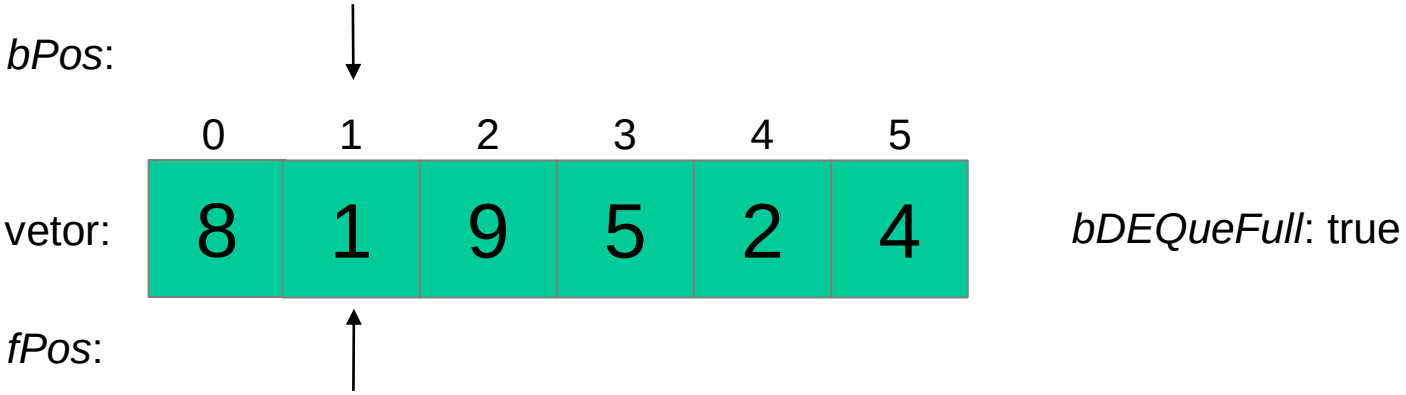
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushFront(1);
```

Coloca o elemento 1

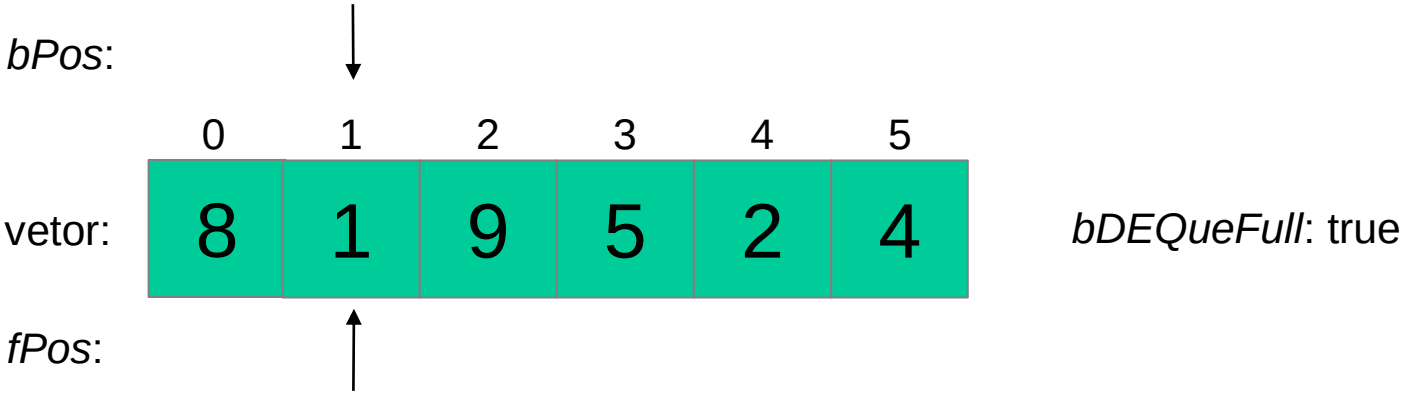
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushFront(7);
```

Verifica se está cheia

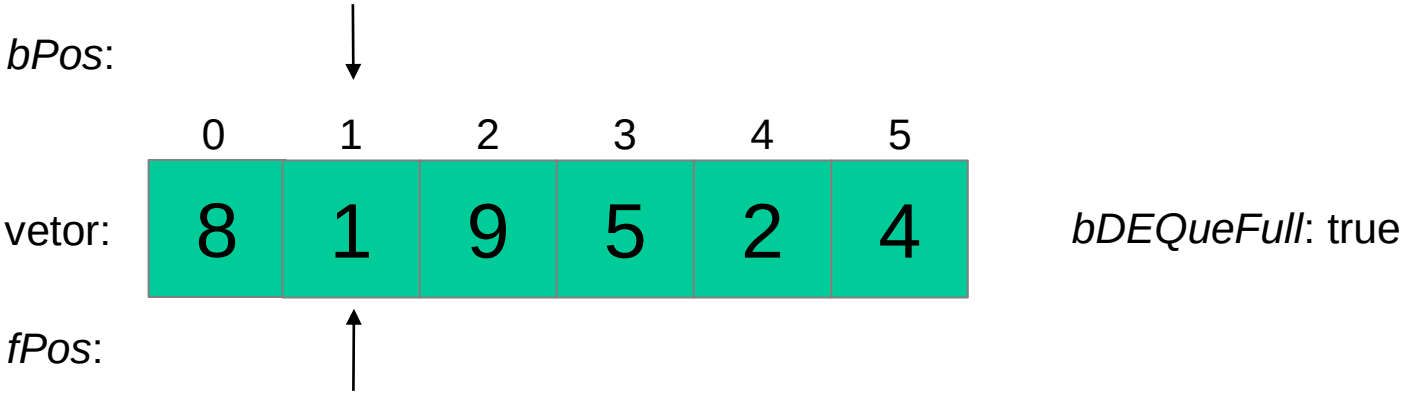
DEQue




```
if ( isOver( ) )
    System.out.println("DEQue Cheia");
else
    pushFront(7);
```

Está cheia! Não coloca o elemento 7

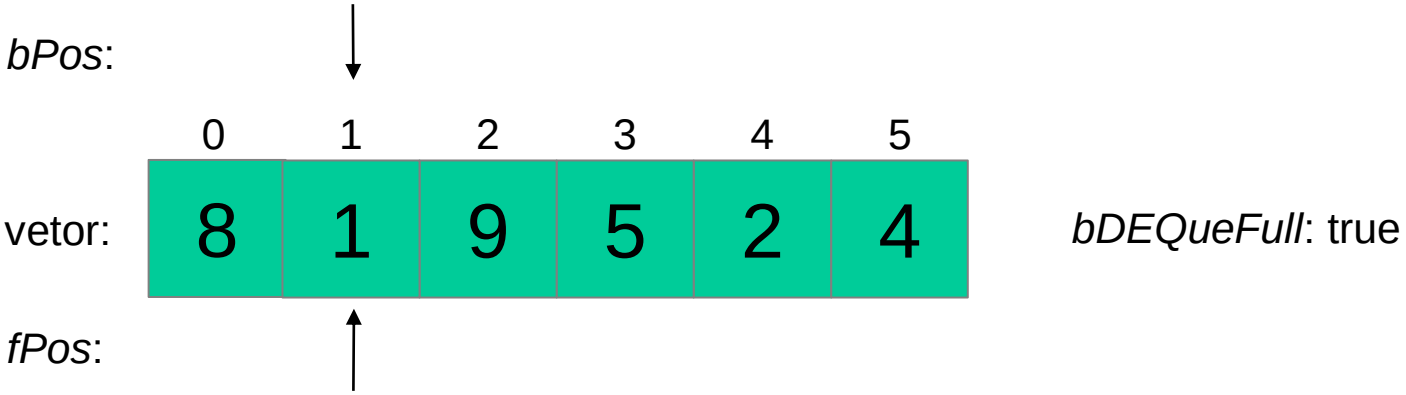
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popBack( );
```

Verifica se está vazia

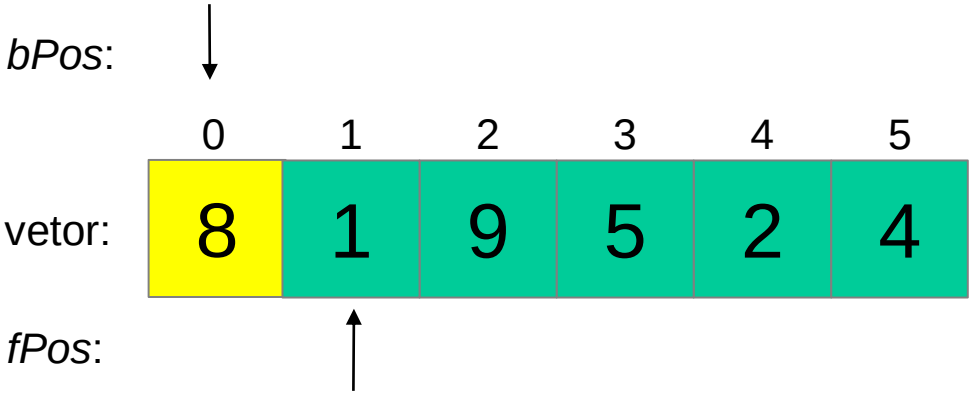
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popBack( );
// n: 8
```

Não está vazia! Retira o elemento de trás

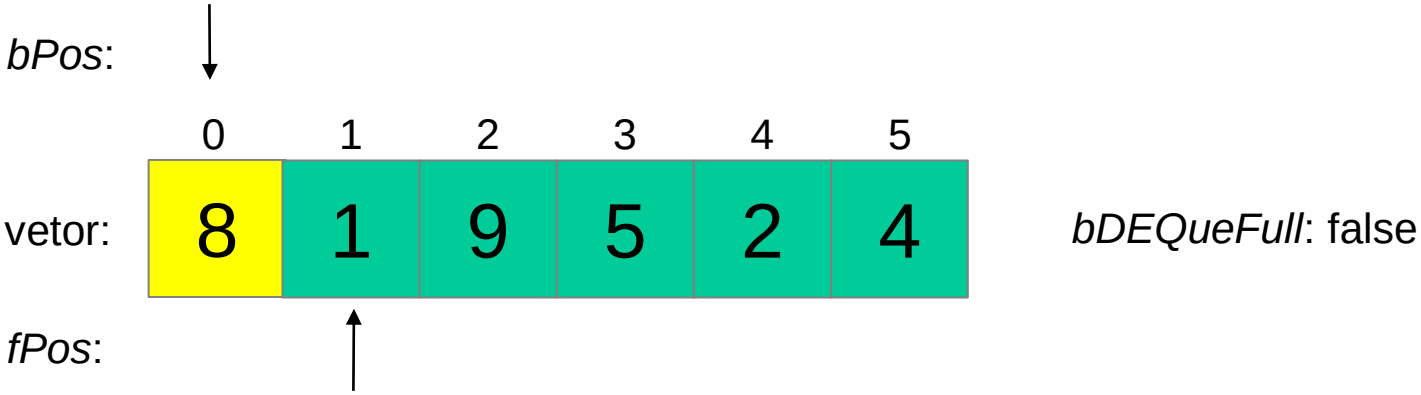
DEQue




```
if ( --bPos < 0 )
    bPos = vetor.length-1;
```

Decrementa bPos (*back*)

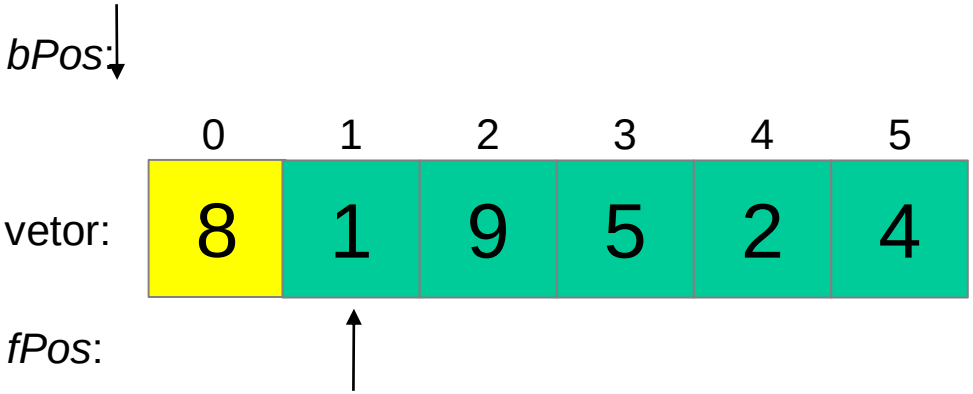
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popBack( );
```

Verifica se está vazia

DEQue

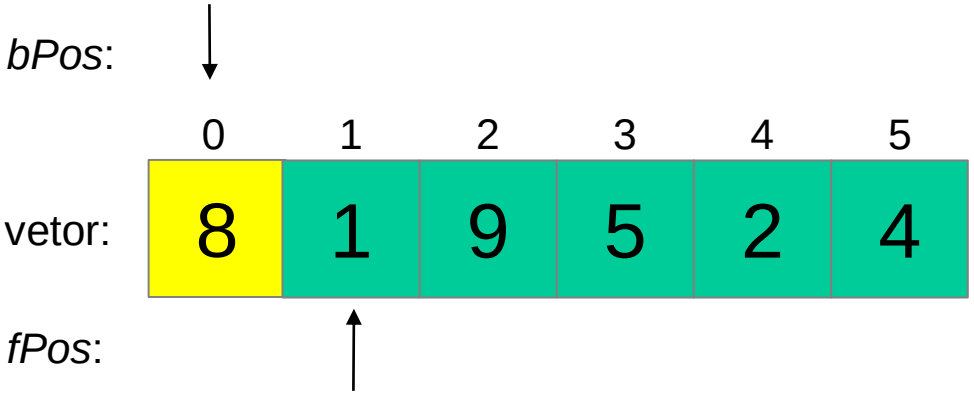


bDEQueFull: false

```
if ( --bPos < 0 )
    bPos = vetor.length-1;
```

Decrementa bPos (*back*)

DEQue

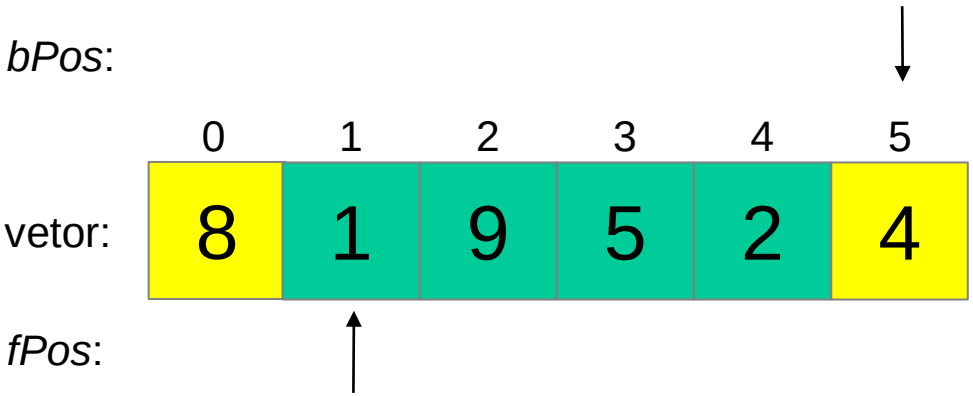


bDEQueFull: false

```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popBack( );
// n: 4
```

Não está vazia! Retira o elemento de trás

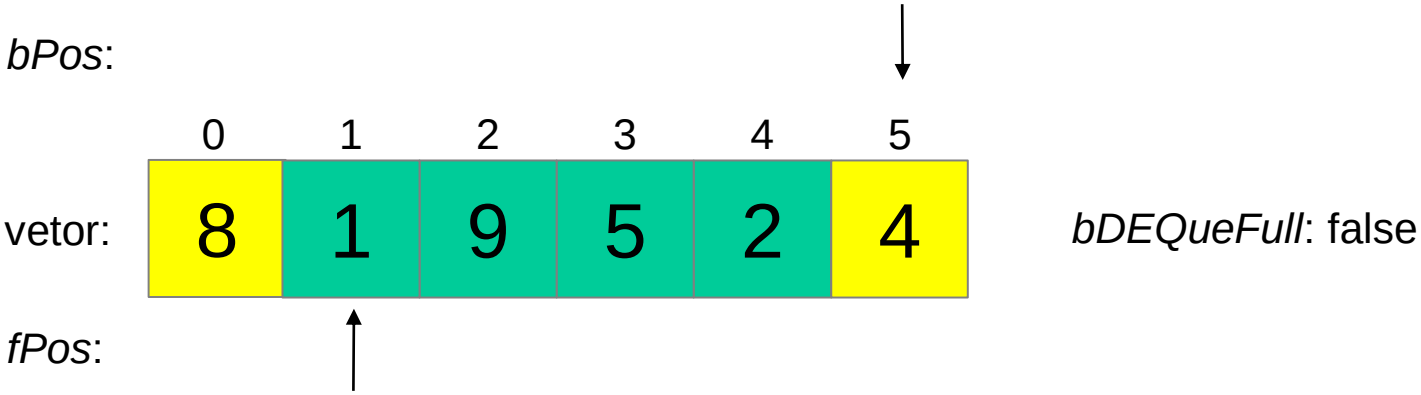
DEQue




```
if ( --bPos < 0 )
    bPos = vetor.length-1;
```

Ajusta bPos (*back*)

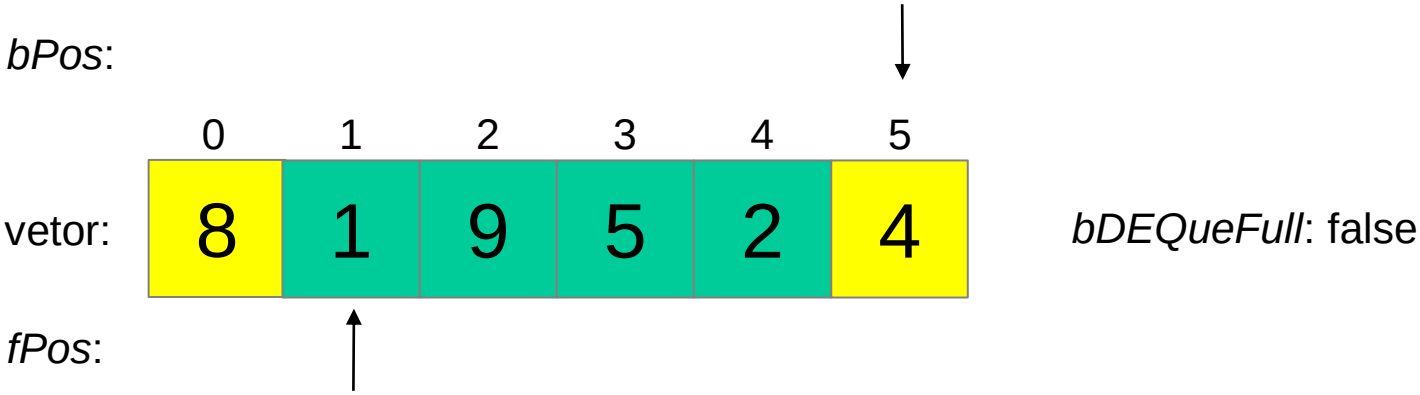
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popBack( );
```

Verifica se está vazia

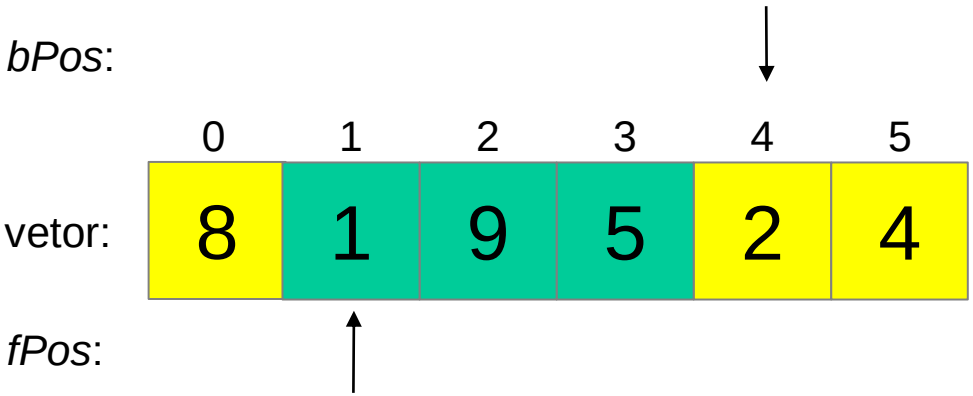
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popBack( );
// n: 2
```

Não está vazia! Retira o elemento de trás

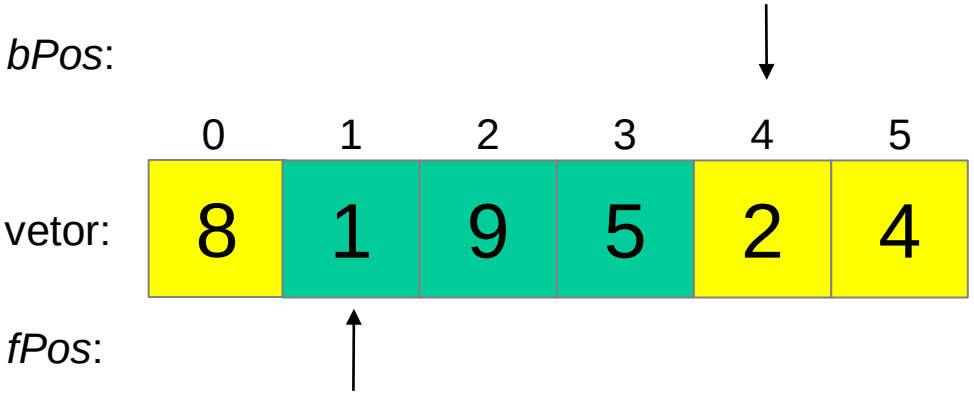
DEQue




```
if ( --bPos < 0 )
    bPos = vetor.length-1;
```

Decrementa bPos (*back*)

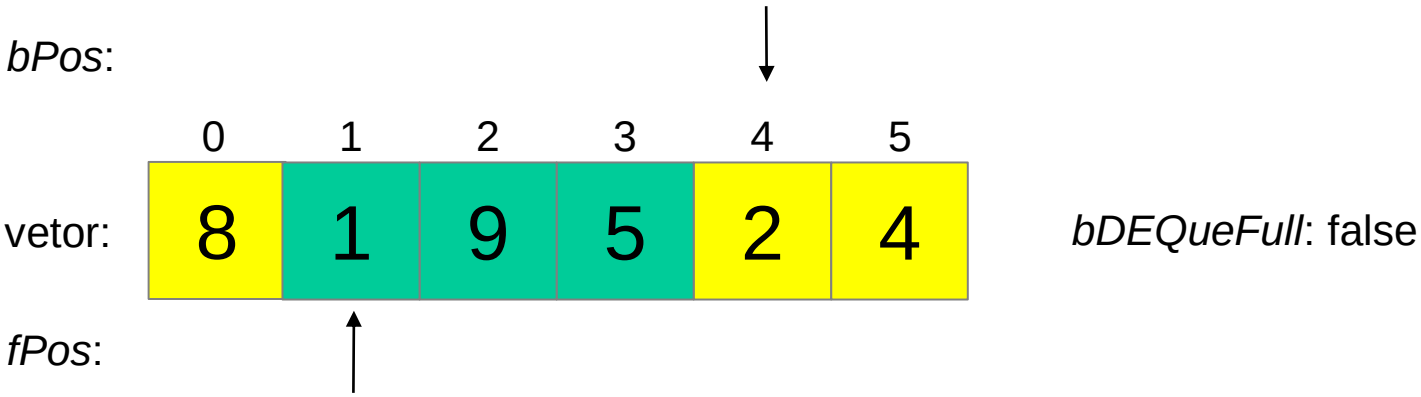
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popBack( );
```

Verifica se está vazia

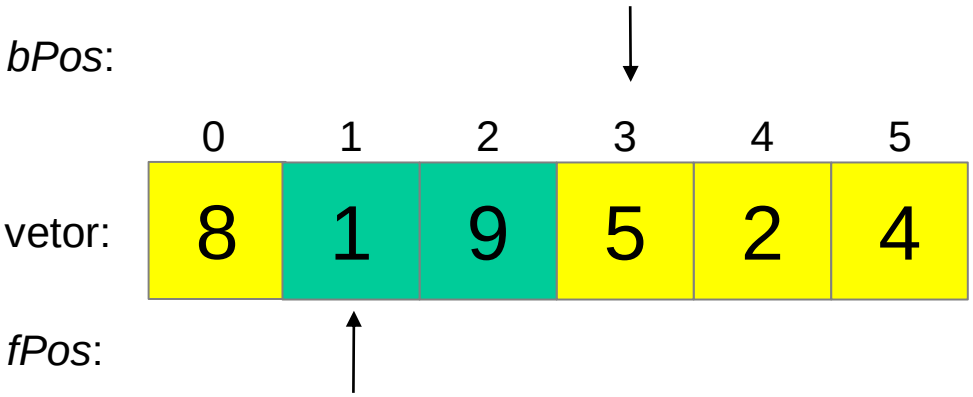
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popBack( );
// n: 5
```

Não está vazia! Retira o elemento de trás

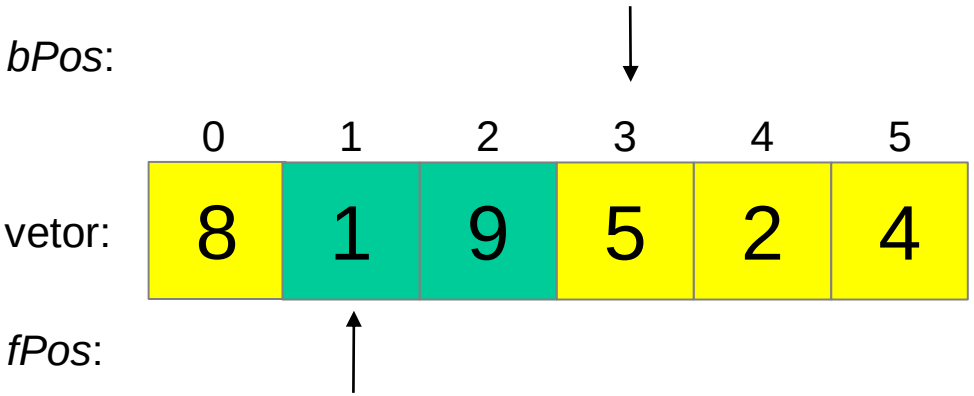
DEQue




```
if ( --bPos < 0 )
    bPos = vetor.length-1;
```

Decrementa bPos (*back*)

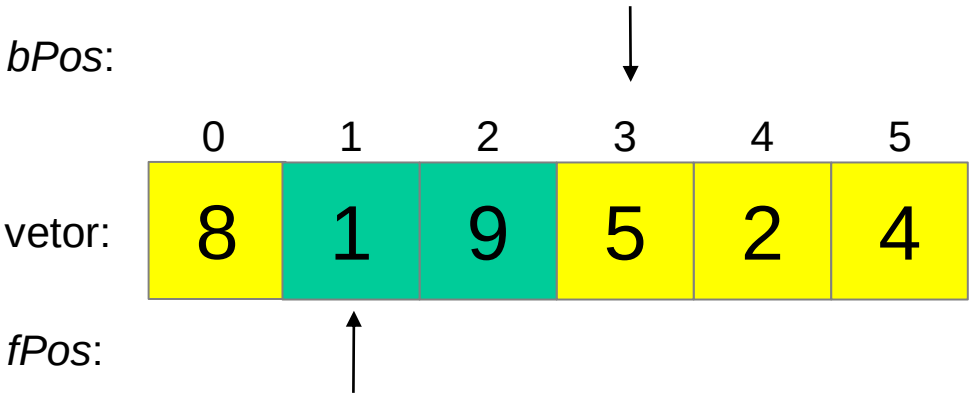
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popFront( );
```

Verifica se está vazia

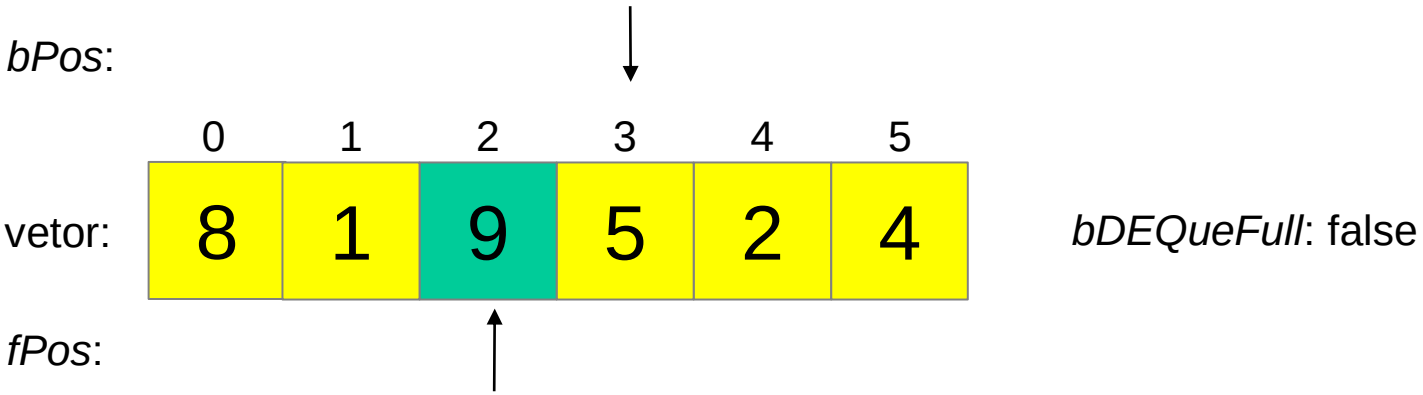
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popFront( );
// n: 1
```

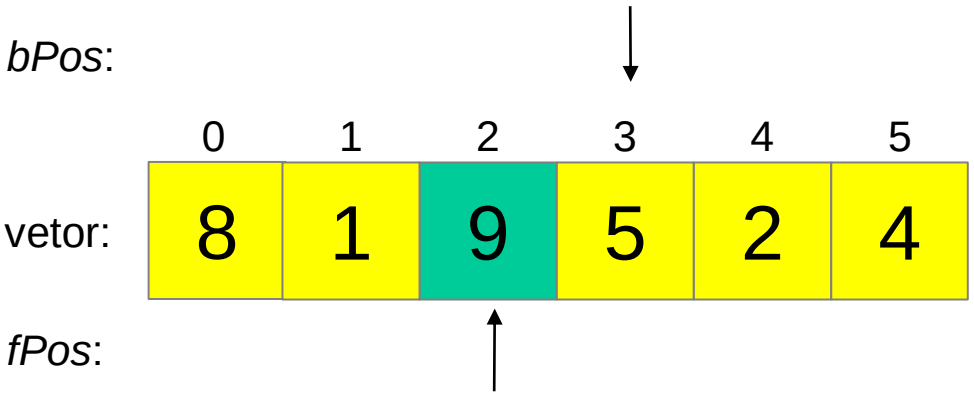
Não está vazia! Retira o elemento da frente

DEQue



Incrementa fPos (*front*)

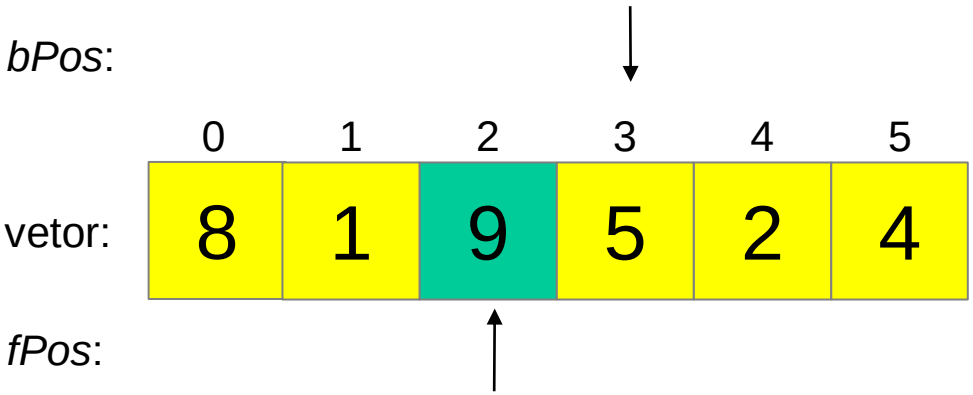
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popFront( );
```

Verifica se está vazia

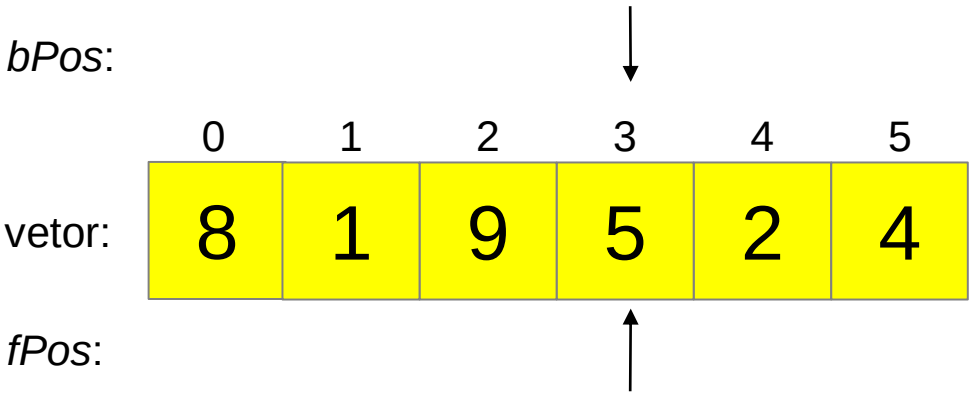
DEQue




```
int n;
:
:
if ( isEmpty( ) )
      System.out.println("DEQue Vazia");
else
       n = popFront( );
// n: 9
```

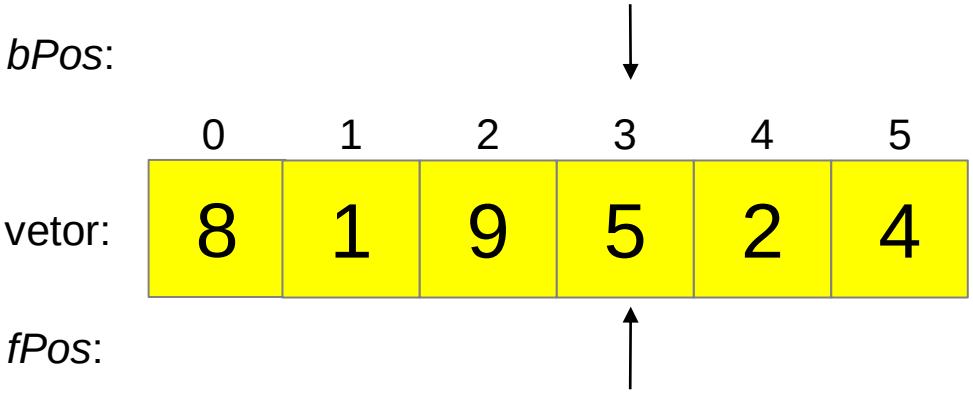
Não está vazia! Retira o elemento da frente

DEQue



Incrementa fPos (*front*)

DEQue



Repete qualquer operação:
pushFront, pushBack, popFront, popBack, size, front, back, isEmpty e isFull
conforme necessidade...

Insere ou retira elementos da *DEQue*...

DEQue

Exemplo de códigos escrito em Java:

Codificação dos Métodos

DEQue

```
public   static    int          iTAM = 10;
public   static    int          iDEQue [ ] = new     int [ iTAM ];
public   static    int          iBPos = 0;
public   static    int          iFPos = 0;
public   static    boolean      bDEQueFull = false;
:
:
:

public   static    int   size( )
{
if( iBPos >= iFPos && !bDEQueFull)      return    iBPos - iFPos;
else      return    iBPos + iDEQue.length - iFPos;
}
```

Método *size( )*

DEQue

```
public   static    int          iTAM = 10;
public   static    int          iDEQue [ ] = new     int [ iTAM ];
public   static    int          iBPos = 0;
public   static    int          iFPos = 0;
public   static    boolean      bDEQueFull = false;
:
:
public   static    int   back( )
{
if( iBPos == 0 )  return    iDEQue[ iDEQue.length – 1 ];
return   iDEQue[ iBPos – 1 ];
}

public   static    int   front( )
{
return   iDEQue[ iFPos ];
}
```

Métodos *back( )* e *front( )*

DEQue

```
public   static    int          iTAM = 10;
public   static    int          iDEQue [ ] = new    int [ iTAM ];
public   static    int          iBPos = 0;
public   static    int          iFPos = 0;
public   static    boolean      bDEQueFull = false;
:
:
public   static    boolean      isOver( )
{
if( iBPos == iFPos && bDEQueFull )      return   true;
return   false;
}
public   static    boolean      isEmpty( )
{
if( iBPos == iFPos && !bDEQueFull )     return   true;
return   false;
}
```

Métodos *isOver( )* e *isEmpty( )*

DEQue

```
public   static    int          iTAM = 10;
public   static    int          iDEQue [ ] = new     int [ iTAM ];
public   static    int          iBPos = 0;
public   static    int          iFPos = 0;
public   static    boolean      bDEQueFull = false;
:
:
:

public   static    int   popBack( )
{
if( --iBPos < 0)  iBPos = iDEQue.length - 1;
bDEQueFull = false;
return   iDEQue[ iBPos ];
}
```

Método *popBack( )*

DEQue

```
public   static    int          iTAM = 10;
public   static    int          iDEQue [ ] = new     int [ iTAM ];
public   static    int          iBPos = 0;
public   static    int          iFPos = 0;
public   static    boolean      bDEQueFull = false;
:
:
:

public   static    int   popFront( )
{
int   iIndice = iFPos++;

if( iFPos >= iDEQue.length )      iFPos = 0;
bDEQueFull = false;
return   iDEQue[ iIndice ];
}
```

Método *popFront( )*

DEQue

```
public   static    int          iTAM = 10;
public   static    int          iDEQue [ ] = new     int [ iTAM ];
public   static    int          iBPos = 0;
public   static    int          iFPos = 0;
public   static    boolean      bDEQueFull = false;
:
:
:

public   static    void      pushBack( int iN )
{
iDEQue[ iBPos++ ] = iN;
if( iBPos >= iDEQue.length )      iBPos = 0;
if( iBPos == iFPos )      bDEQueFull = true;
}
```

Método *pushBack( )*

DEQue

```
public   static    int          iTAM = 10;
public   static    int          iDEQue [ ] = new     int [ iTAM ];
public   static    int          iBPos = 0;
public   static    int          iFPos = 0;
public   static    boolean      bDEQueFull = false;
:
:
:

public   static    void      pushFront( int iN )
{
if( --iFPos < 0 ) iFPos = iDEQue.length - 1;
if( iFPos == iBPos )       bDEQueFull = true;
iDEQue[ iFPos ] = iN;
}
```

Método *pushFront( )*

DEQue

FIM