ESTRUTURAS DE DADOS

2024/2025

Aula 05

- Colecção *Queue*
- Implementação em Lista Ligada
- Implementação em *array*
- Implementação em *array* circular

Ricardo Santos | rjs@estg.ipp.pt

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Queues

- Uma fila (***Queue***) é uma colecção cujos elementos são adicionados numa extremidade e removidos da outra

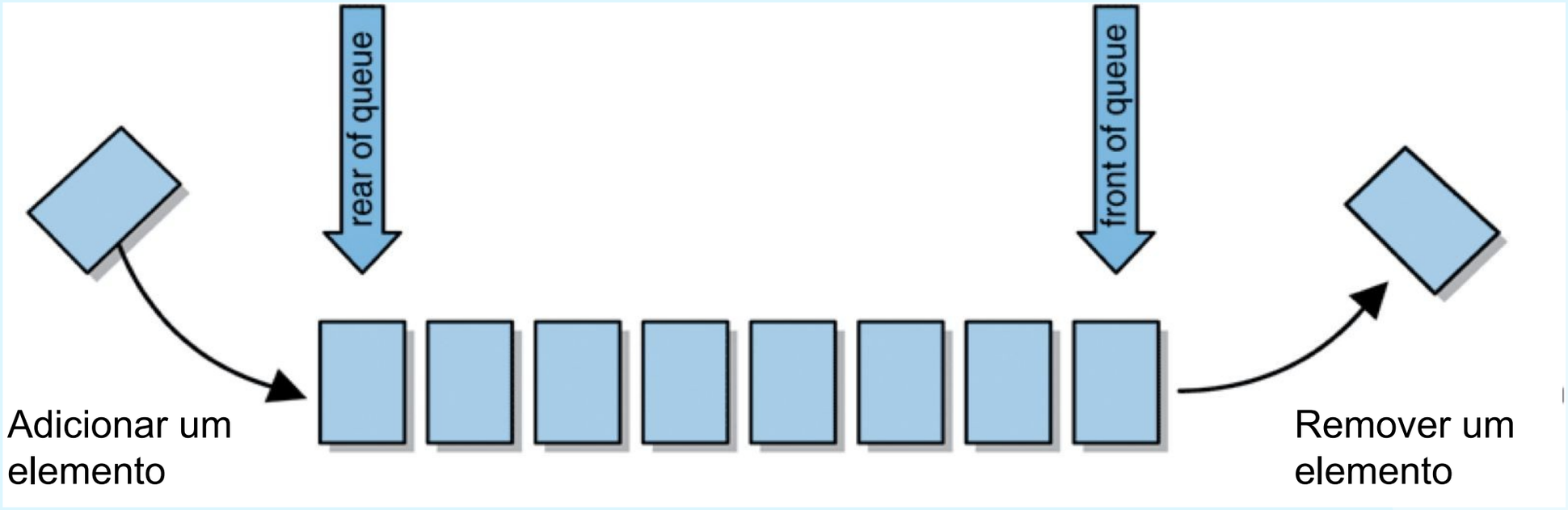
- Portanto, uma fila (***Queue***) é processada de forma **FIFO: first in, first out**

- Os elementos são retirados na mesma ordem em que chegam

- Qualquer fila de espera é uma fila (***Queue***):
  - o *check-out* de uma fila de supermercado
  - os carros num semáforo
  - uma linha de montagem

- Uma fila é normalmente representada na horizontal
- Uma extremidade da fila é a parte traseira (ou cauda), onde os elementos são adicionados
- O outro lado é a frente (ou cabeça), a partir da qual os elementos são removidos
- Ao contrário de uma pilha (***Stack***), que opera no final da colecção, uma fila opera em ambas as extremidades

Vista Conceptual de uma *Queue*



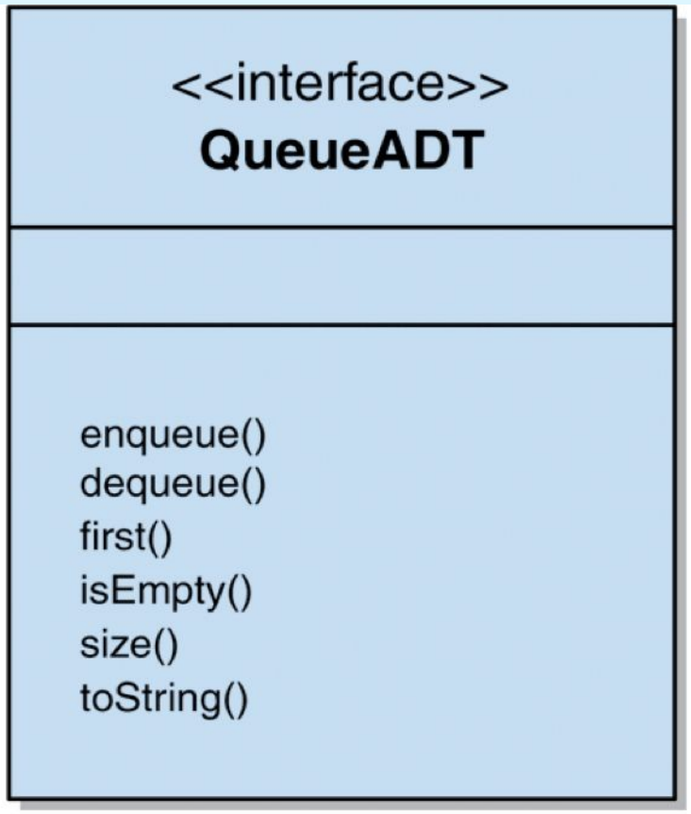
Operações de uma *Queue*

- O termo enqueue é usado para se referir ao processo de adicionar um elemento na fila (***Queue***)
- Da mesma forma, retirar da fila (`dequeue`) é o processo de remoção de um elemento
- Como uma pilha, um fila não permite aos utilizadores aceder aos elementos no meio da fila
- Incluímos um método `toString` apenas por conveniência

Operações da Queue

Operação	Descrição
`enqueue`	Adiciona um elemento à traseira da fila
`dequeue`	Remove o elemento da frente da fila
`first`	Examina o elemento na frente da fila
`isEmpty`	Determina se a fila está vazia
`size`	Determina o número de elementos da fila
`toString`	Representação da fila em string

Interface `QueueADT`



Interface QueueADT

```
public interface QueueADT<T> {

    /**
     * Adds one element to the rear of this queue.
     *
     * @param element  the element to be added to
     * the rear of this queue
     */
    public void enqueue(T element);

    /**
     * Removes and returns the element at the front of
     * this queue.
     *
     * @return  the element at the front of this queue
     */
    public T dequeue();

    /**
     * Returns without removing the element at the front of
     * this queue.
     *
     * @return  the first element in this queue
     */
    public T first();
```

```
    /**
     * Returns true if this queue contains no elements.
     *
     * @return  true if this queue is empty
     */
    public boolean isEmpty();

    /**
     * Returns the number of elements in this queue.
     *
     * @return  the integer representation of the size
     * of this queue
     */
    public int size();

    /**
     * Returns a string representation of this queue.
     *
     * @return  the string representation of this queue
     */
    public String toString();
}
```

Exemplo: Mensagens Codificadas

- Vamos usar uma fila para nos ajudar a codificar e descodificar mensagens
- A cifra de César codifica uma mensagem deslocando cada letra numa mensagem por um valor constante **k**
- Se **k** é **5**, **A** torna-se **F**, **B** torna-se **G**, etc
- No entanto, isso é bastante fácil de descodificar
- Uma melhoria pode ser feita alterando o quanto uma letra é mudada dependendo do local onde a letra está na mensagem

- A chave de repetição é uma série de números inteiros que determina quanto é que cada caracter será movido

- Por exemplo, considere a chave de repetição

```
3 1 7 4 2 5
```

- O primeiro caracter da mensagem é deslocado **3** posições, o **7** seguinte, e assim por diante

- Quando a chave está esgotada, é recomeçado do início da chave

Mensagem codificada com recurso a uma chave de repetição

Mensagem Codificada:	n	o	v	a	n	j	g	h	l		m	u		u	r	x	l	v
Chave:	3	1	7	4	2	5	3	1	7		4	2		5	3	1	7	4
Mensagem Descodificada:	k	n	o	w	l	e	d	g	e		i	s		p	o	w	e	r

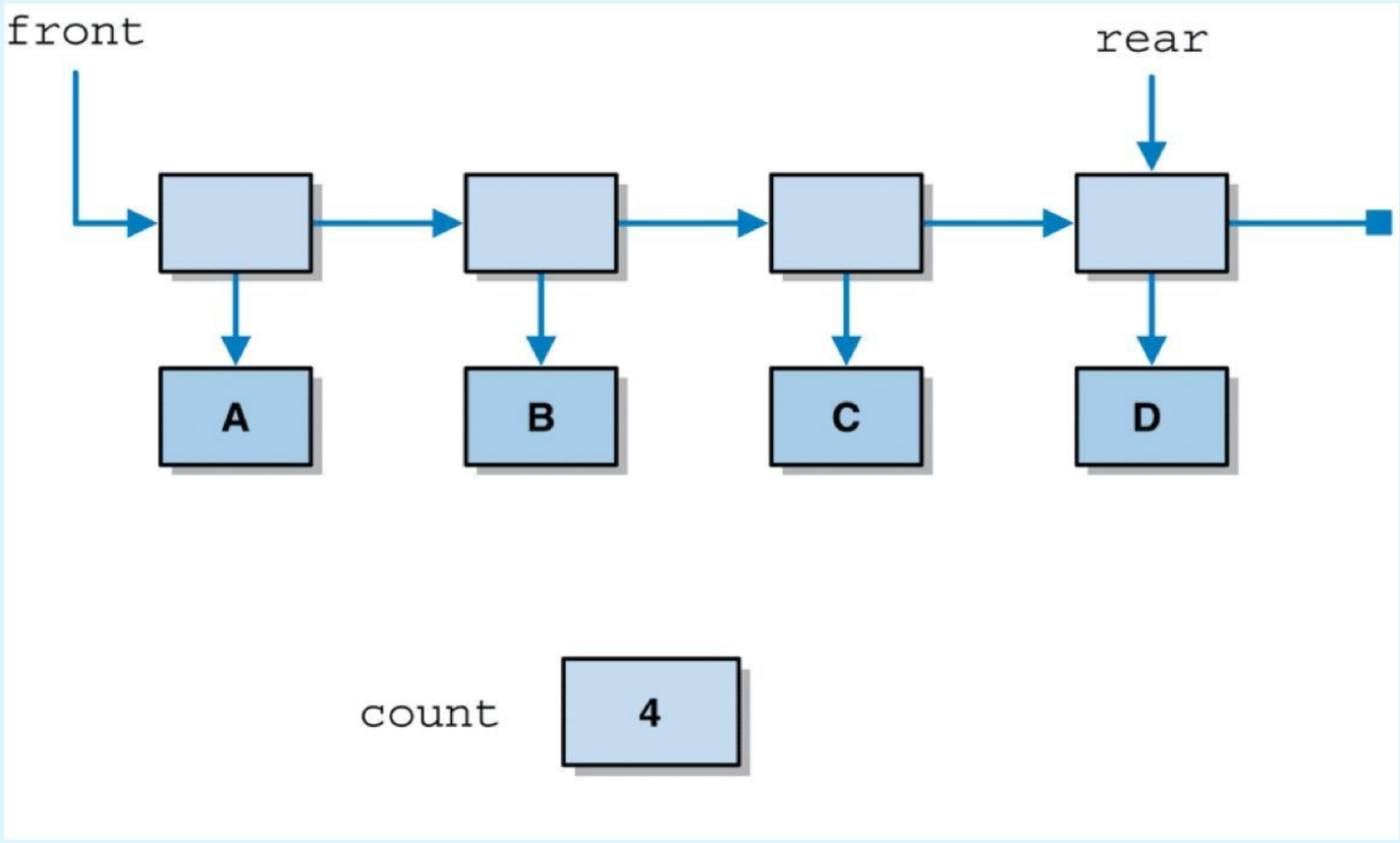
- **Exercício:** Implementar um programa para a codificação de mensagens com recurso a uma ***Queue***

Implementar a *Queue* com recurso a uma Lista Ligada

Classe `LinkedQueue`

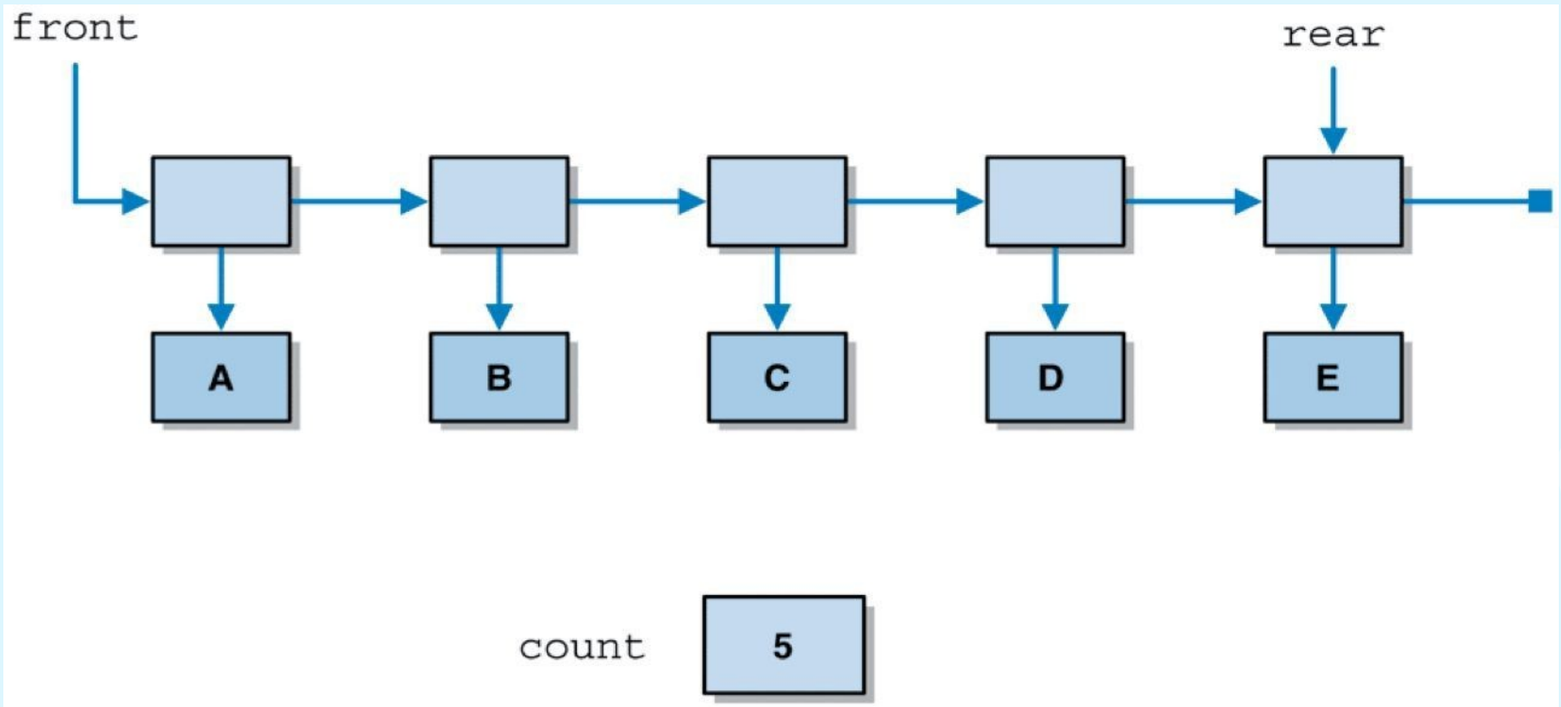
- Tal como uma pilha (***Stack***), uma fila (***Queue***) pode ser implementada usando um array ou uma lista ligada
- A versão com recurso a uma lista ligada pode usar a classe `LinearNode` novamente
- Além de manter uma referência no início da lista, vamos manter uma segunda referência para o fim
- Um número inteiro vai contar o número de elementos na fila

Queue Inicial

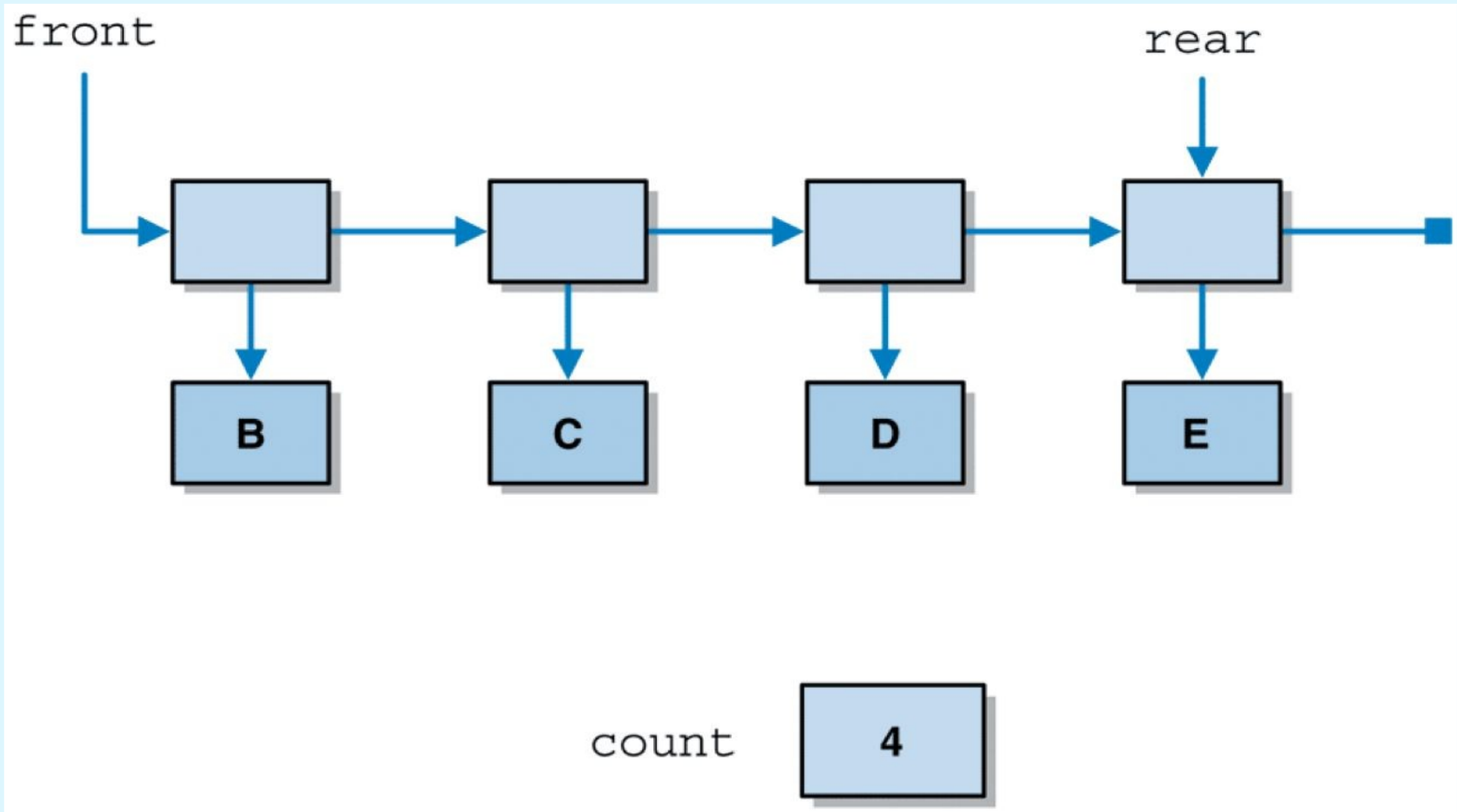


- **Exercício:** Implementar a `LinkedQueue`

LinkedQueue: **Operação** enqueue



LinkedQueue: **Operação** dequeue



`LinkedQueue`:
Outras operações

- As restantes operações `LinkedQueue` são relativamente simples e são semelhantes às da colecção `LinkedStack`

- A operação `first` é executada, retornando uma referência para o elemento na frente da fila

- A operação `isEmpty` retorna verdadeiro se a contagem dos elementos é `0`, e falso caso contrário

- A operação `size` simplesmente retorna a contagem de elementos na fila
- O método `toString` retorna uma *string* composta pelos resultados de `toString` de cada elemento

Agora vamos implementar uma *Queue* recorrendo a um *array*

Implementar *Queues* com *arrays*

- Uma estratégia para implementar uma ***Queue*** seria a de fixar uma ponta da ***Queue*** na posição `0` do *array*, como fizemos com o `ArrayStack`

- As ***Queues***, operam em ambas as extremidades, o que implica o deslocamento de elementos, tanto no enqueue como dequeue

- Uma abordagem melhor é a de usar um *array* circular e não fixar qualquer ponta da fila

Classe `CircularArrayQueue`

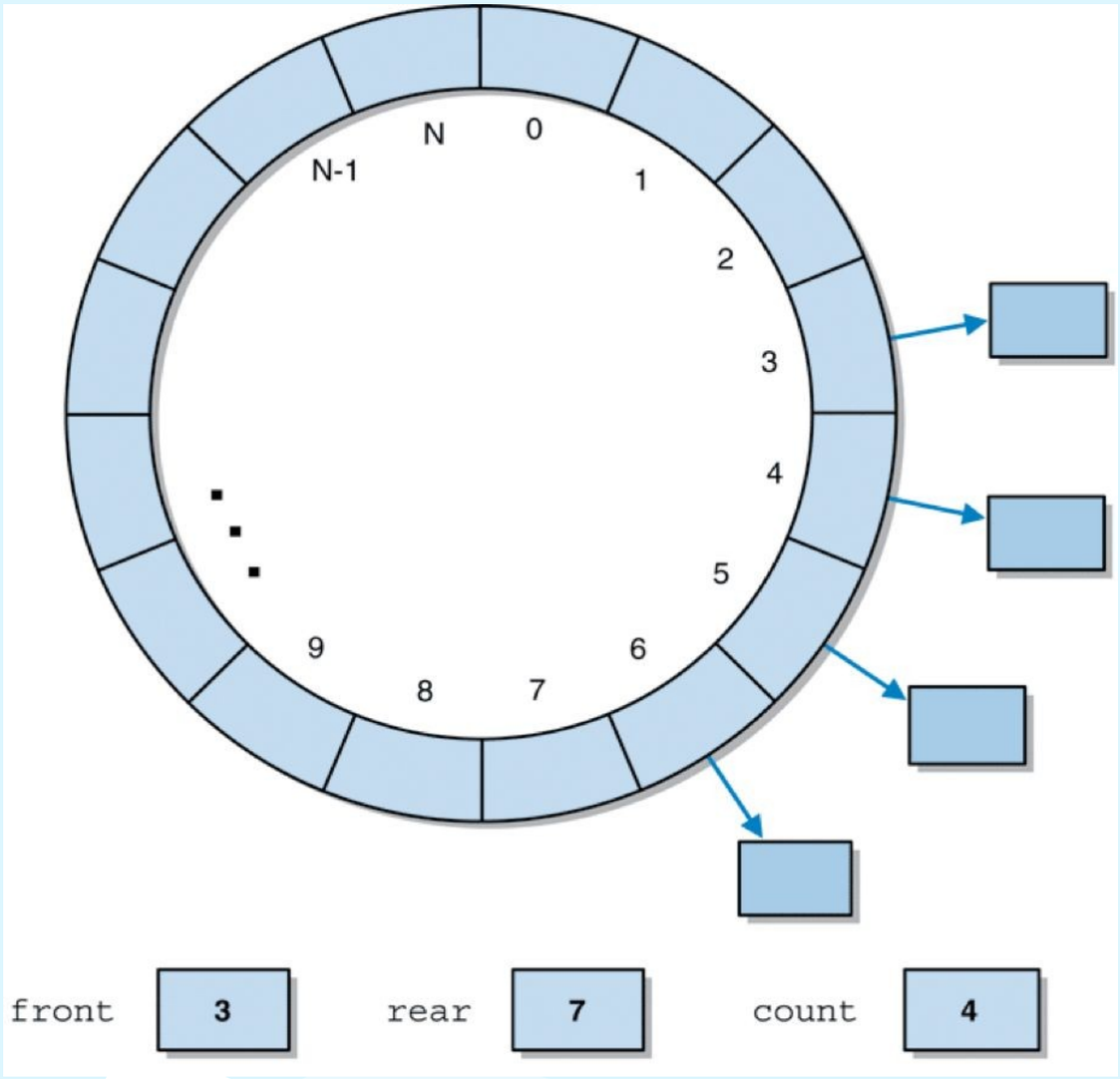
- Se não fixarmos qualquer uma das pontas da ***Queue*** no índice `0`, não teremos que mudar os elementos

- A ***Queue*** circular é uma implementação de uma ***Queue*** usando um *array* que, conceptualmente realiza ciclos em torno de si

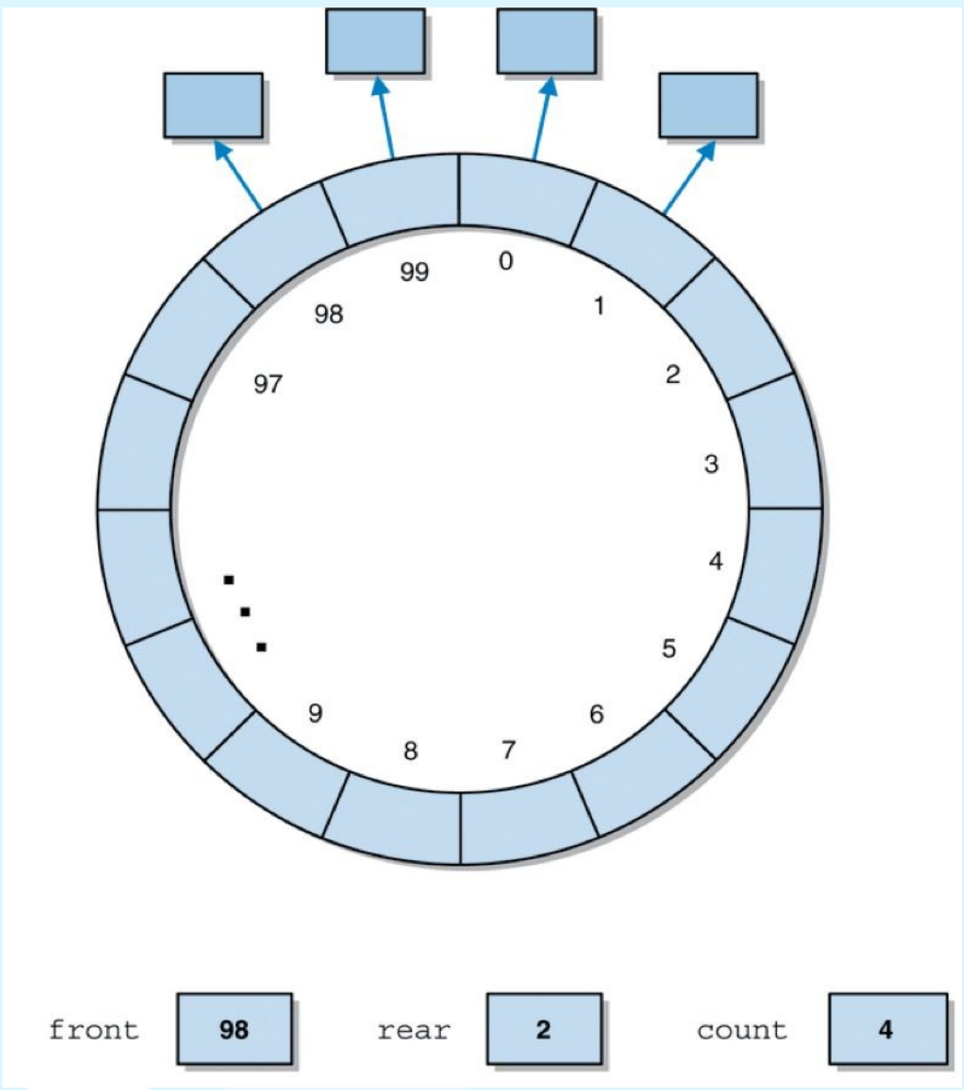
- Ou seja, o último índice é pensado para preceder o índice `0`

- Para isso acompanhamos dois inteiros que indicam onde a frente e a traseira da fila estão, em determinado momento

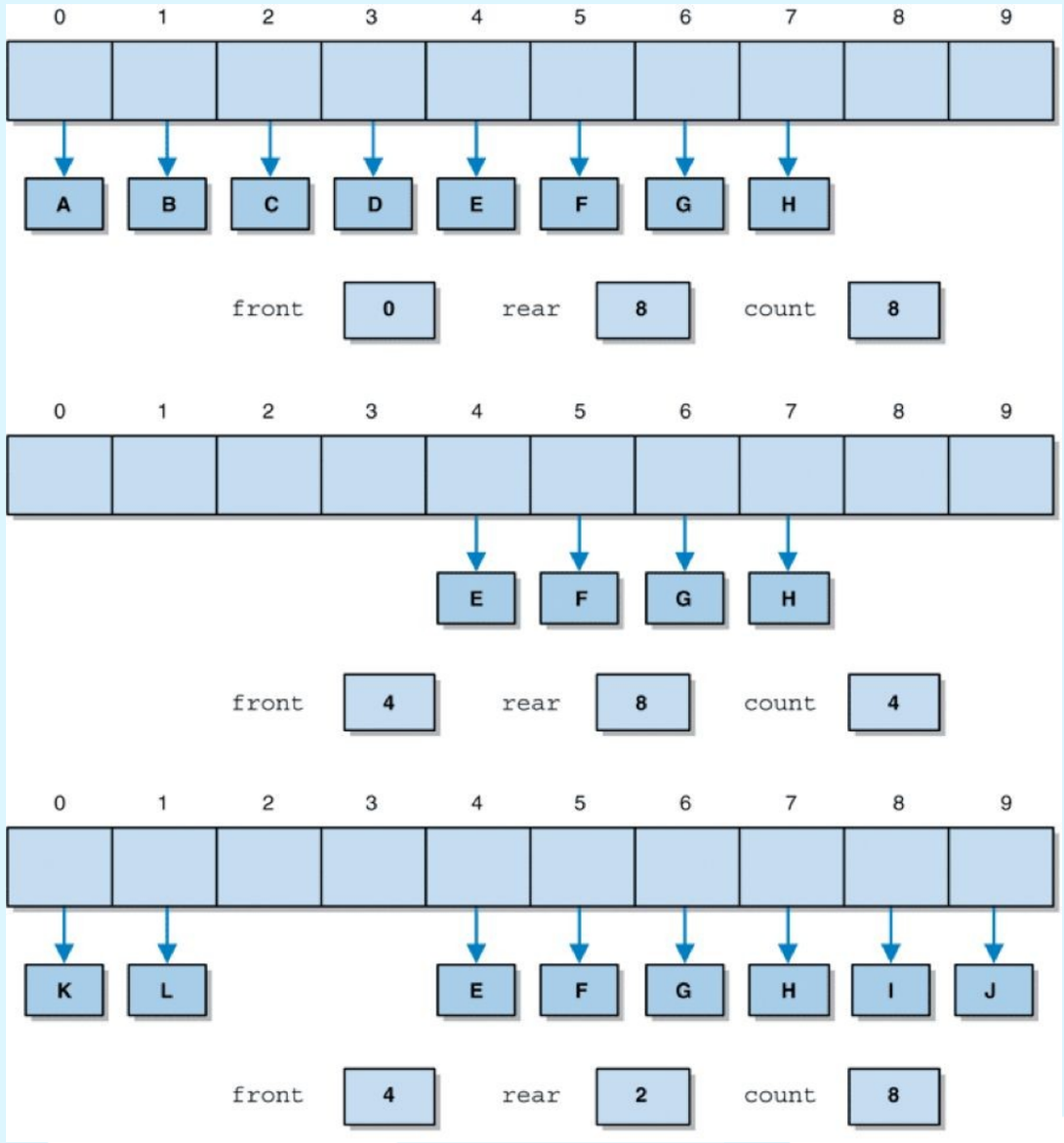
Implementação de uma *Queue* com *array* circular



Uma *Queue* abrangendo o fim de uma *array* circular



Alterações no *array* circular de uma *Queue*



Queues Circulares

- Quando realizamos o enqueue de um elemento, o valor da cauda é incrementado
- É necessário ter em conta a necessidade de voltar à posição 0

```
rear = (rear+1) % queue.length;
```

- Note que esta implementação de array pode também esgotar a capacidade e ser necessário aumentar o tamanho

- **Exercício:** Implementar a `CircularArrayQueue`

Implementações da *Queue*

- A operação `enqueue` é **O(1)** para ambas as implementações
- A operação `dequeue` é **O(1)** para a ***Queue*** em lista ligada e em *array* circular, mas é **O(n)** para a versão em *array* não circular devido à necessidade de fazer um *shift* aos elementos da ***Queue***