ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS LINEALES

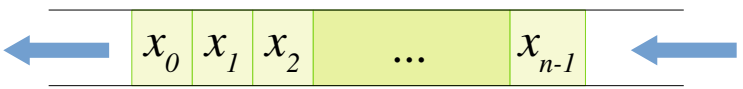
Implementando el TAD Cola

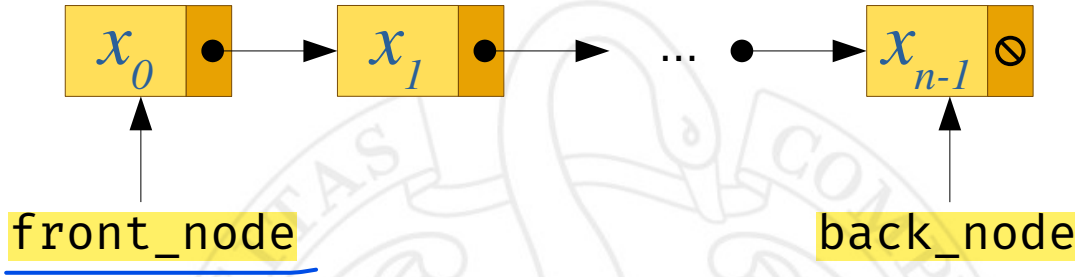
Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Vamos a presentar dos implementaciones: Listas enlazadas y vectores circulares.

Implementación mediante listas enlazadas

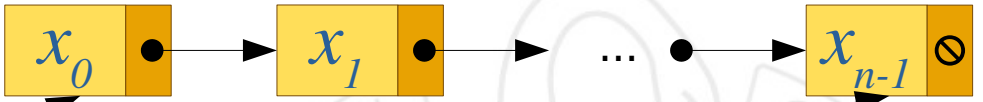
Implementación mediante listas enlazadas





Clase QueueLinkedList

```
template<typename T>
class QueueLinkedList {

   ...

private:
  struct Node {
    T value;
    Node *next;
  };

  Node *front_node;
  Node *back_node;
};
```



- Si la cola está vacía:

  `front_node = back_node = nullptr`

Interfaz pública de QueueLinkedList

```
template<typename T>
class QueueLinkedList {

  QueueLinkedList();
  QueueLinkedList(const QueueLinkedList &other);
  ~QueueLinkedList();

  QueueLinkedList & operator=(const QueueLinkedList &other);

  void push(const T &elem);
  void pop();
  T & front();
  const T & front() const;
  bool empty() const;

  ...

};
```

sobrecarga del operador equals.

encolar

desencolar

Devuelve el primer elemento de la lista.

Devuelve si es una lista vacía.

Interfaz pública de QueueLinkedList

```
template<typename T>
class QueueLinkedList {

  QueueLinkedList();
  QueueLinkedList(const QueueLinkedList &other);
  ~QueueLinkedList();

  QueueLinkedList & operator=(const QueueLinkedList &other);

  void push(const T &elem);
  void pop();
  T & front();
  const T & front() const;
  bool empty() const;

   ...

};
```

Vamos a ver las implementaciones de estos métodos. el T&front se implementaría de manera muy parecida al const T & front()

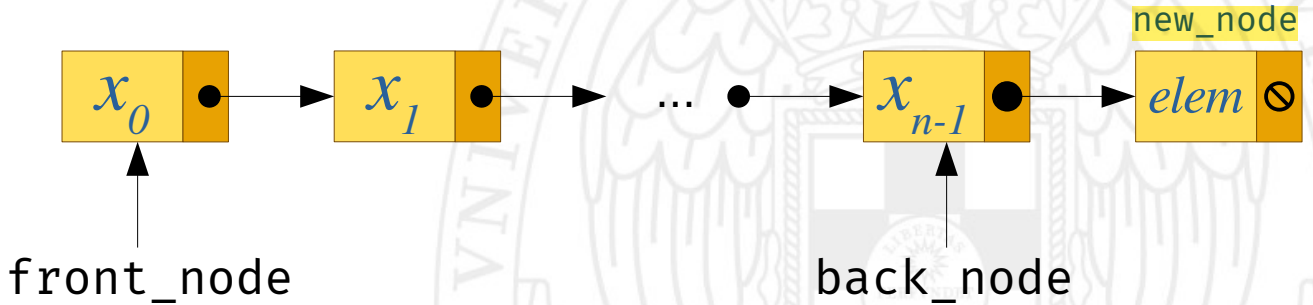
Método push( )

Metemos elemento al final de la lista.

```
void push(const T &elem) {
  Node *new_node = new Node { elem, nullptr };
  if (back_node == nullptr) {
    back_node = new_node;
    front_node = new_node;
  } else {
    back_node→next = new_node;
    back_node = new_node;
  }
}
```

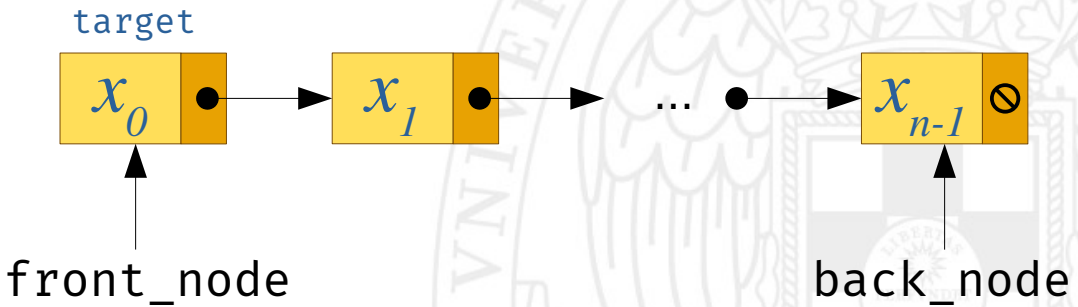
caso de que la lista sea vacía.

Lo añadimos al final de la lista.



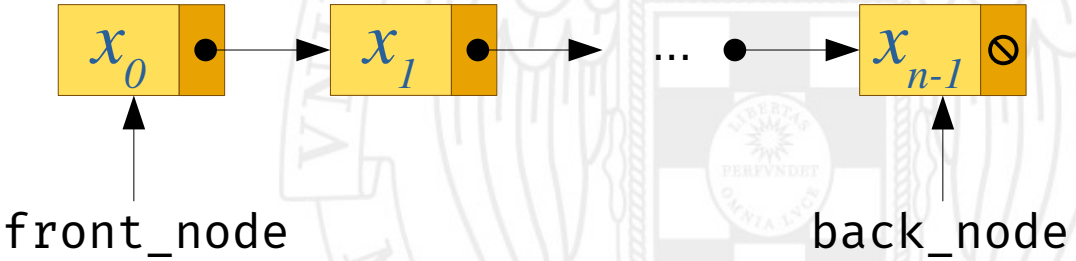
Método pop( )

```
void pop() {
  assert (front_node ≠ nullptr);
  Node *target = front_node;
  front_node = front_node→next;
  if (back_node == target) {
    back_node = nullptr;
  }
  delete target;
}
```



Métodos front() y empty()

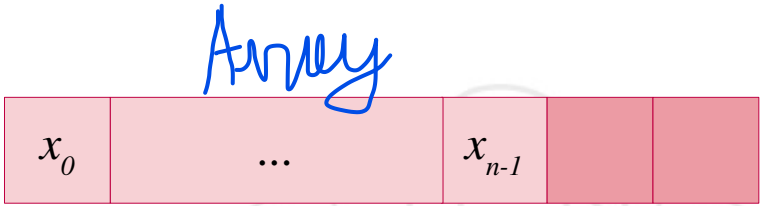
```
const T & front() const {
  assert (front_node ≠ nullptr);
  return front_node→value;
}

bool empty() const {
  return (front_node == nullptr);
}
```



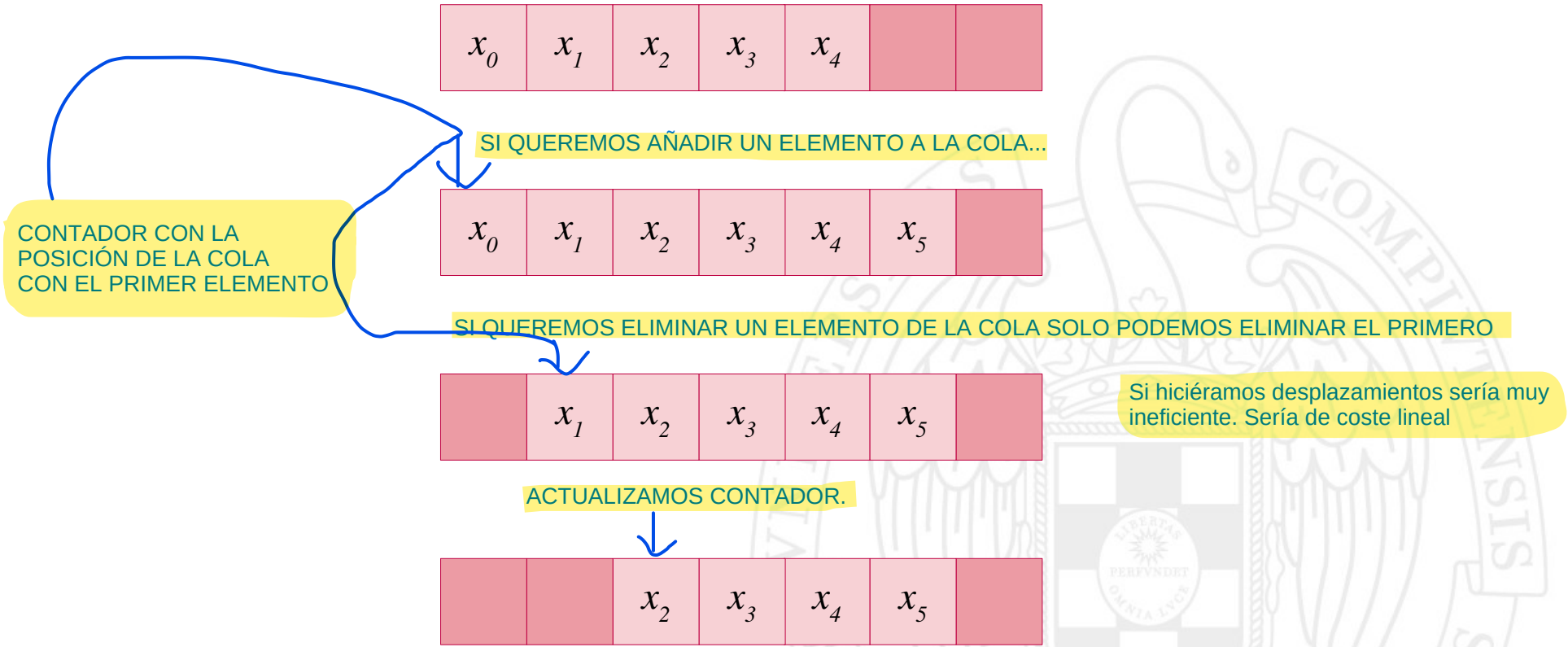
Implementación mediante vectores circulares

Los llamamos vectores circulares por la forma en la que los vamos a utilizar.

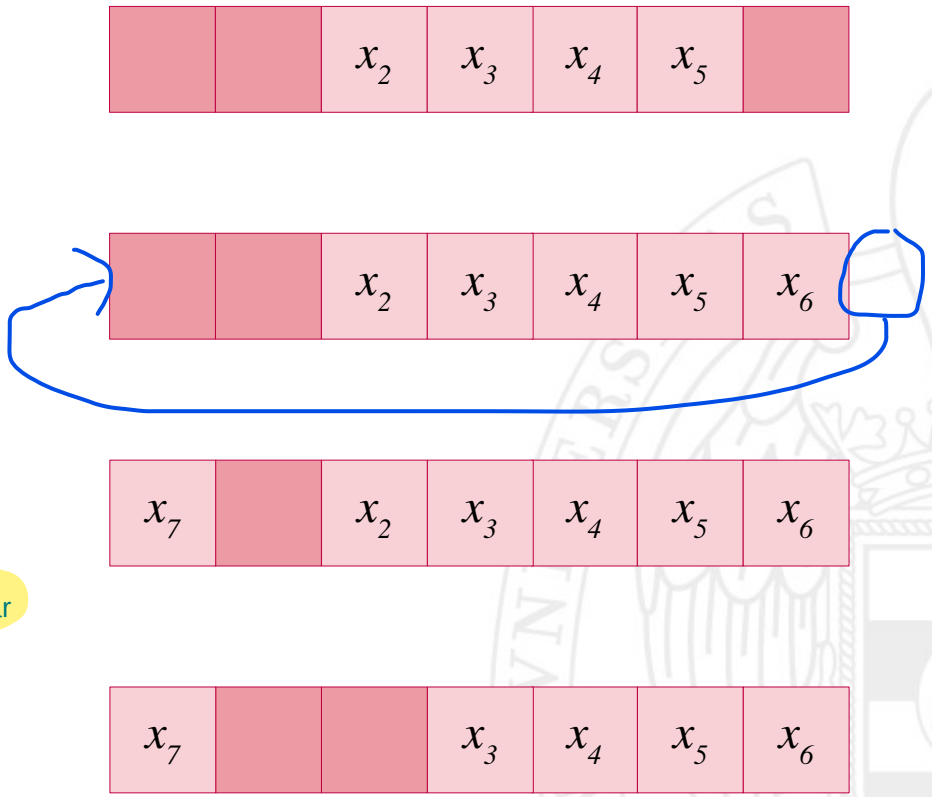
Implementación mediante vectores circulares



Idea: vectores circulares



Idea: vectores circulares



Vector circular

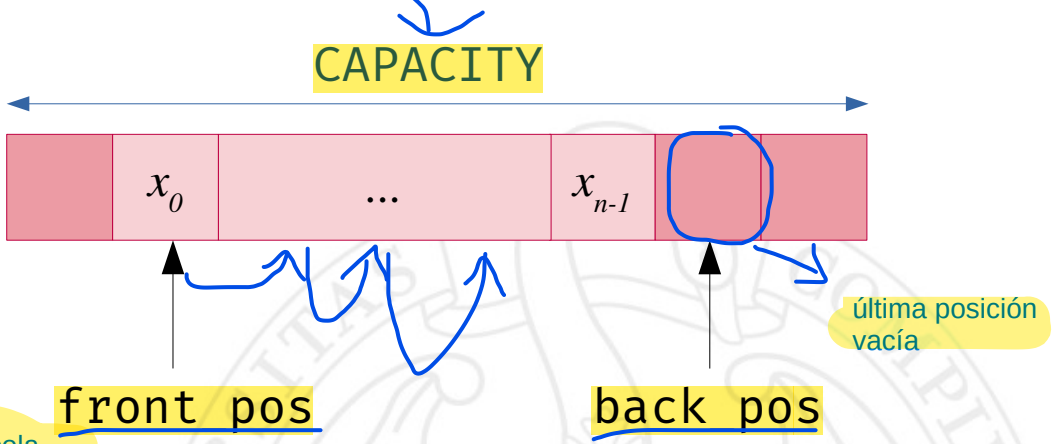
Clase QueueArray

const int CAPACITY = 100

```
class QueueArray {
public:

   ...

private:
   T *elems;
   int front_pos, back_pos;

};
```

posición del primer elemento de la cola

posición del último elemento de la cola.



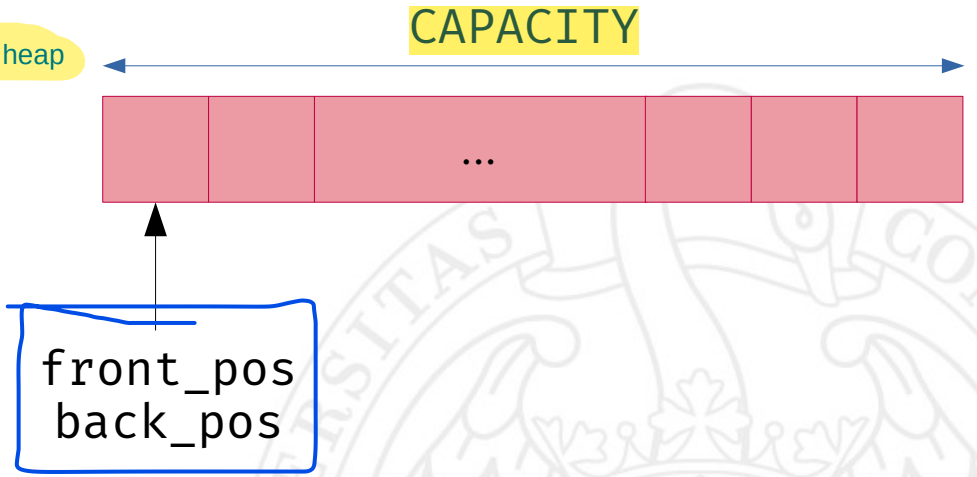
- Si la cola está vacía:

  `front_pos == back_pos`

Constructor

```
QueueArray() {
  elems = new T[CAPACITY];
  front_pos = 0;
  back_pos = 0;
}
```

inicialización en el heap



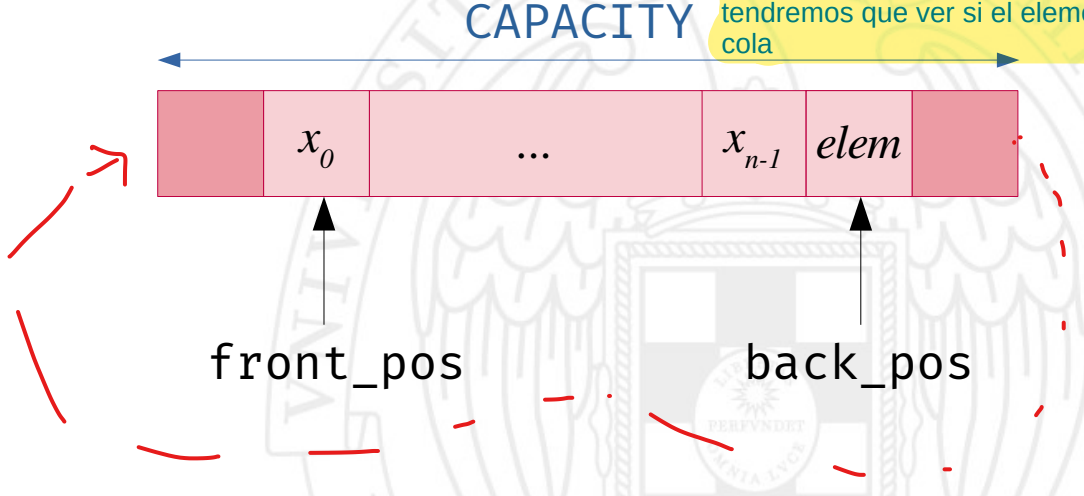
Método push( )

```
void push(const T &elem) {
  // Cabe el elemento en la cola?
  assert ((back_pos + 1) % CAPACITY ≠ front_pos);
  elems[back_pos] = elem;
  back_pos = (back_pos + 1) % CAPACITY;
}
```

Array lleno

Para que vuelva a la posición 0 en el caso de que vuelva por atrás.

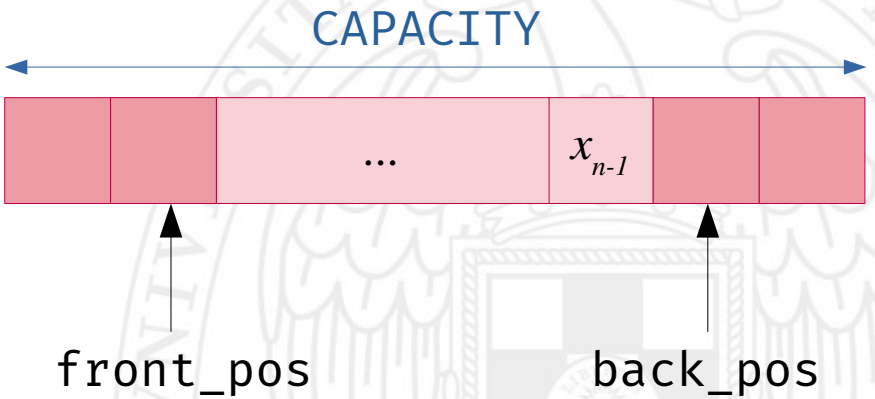
array es de longitud fija en este caso, tendremos que ver si el elemento cabe en la cola



Método pop( )

Eliminar el primer elemento de la lista.

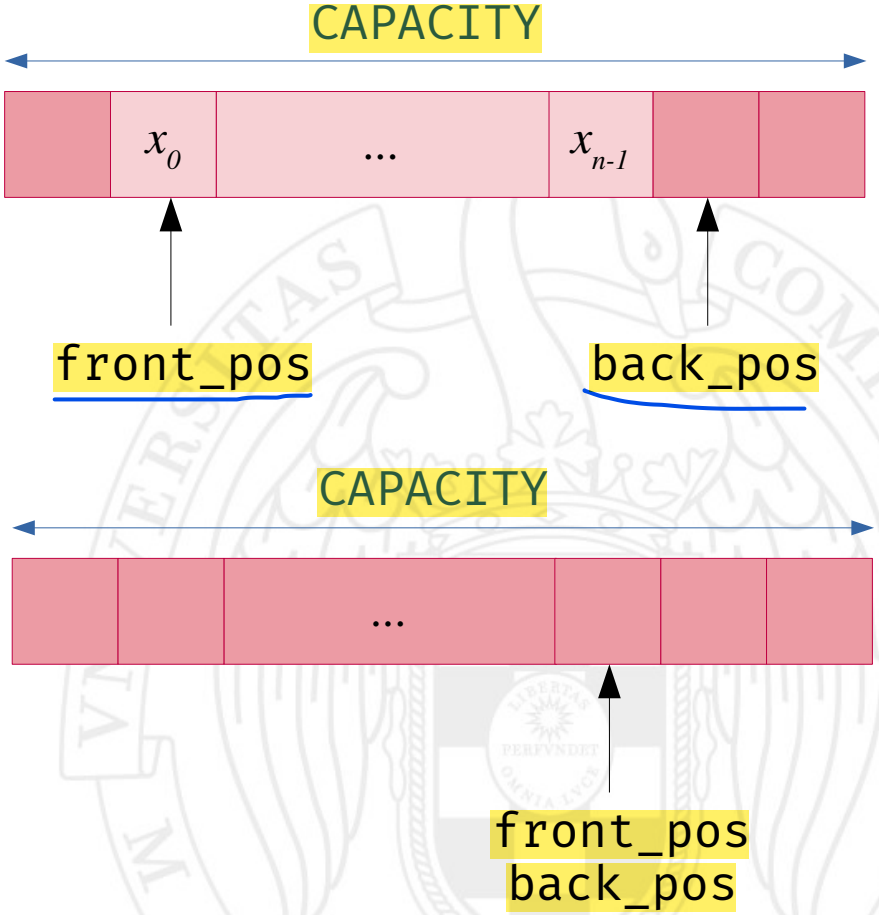
```
void pop() {
  assert (front_pos ≠ back_pos);
  front_pos = (front_pos + 1) % CAPACITY;
}
```



Métodos front() y empty()

```
const T & front() const {
  assert (front_pos ≠ back_pos);
  return elems[front_pos];
}
```

```
bool empty() const {
  return front_pos == back_pos;
}
```



Coste de las operaciones

Operación	Listas enlazadas	Vectores circulares
`push`	$O(1)$	$O(1)$
`pop`	$O(1)$	$O(1)$
`front`	$O(1)$	$O(1)$
`empty`	$O(1)$	$O(1)$

n = número de elementos en la cola