UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Welcome to
Algorithms and Data Structures!
CS2100

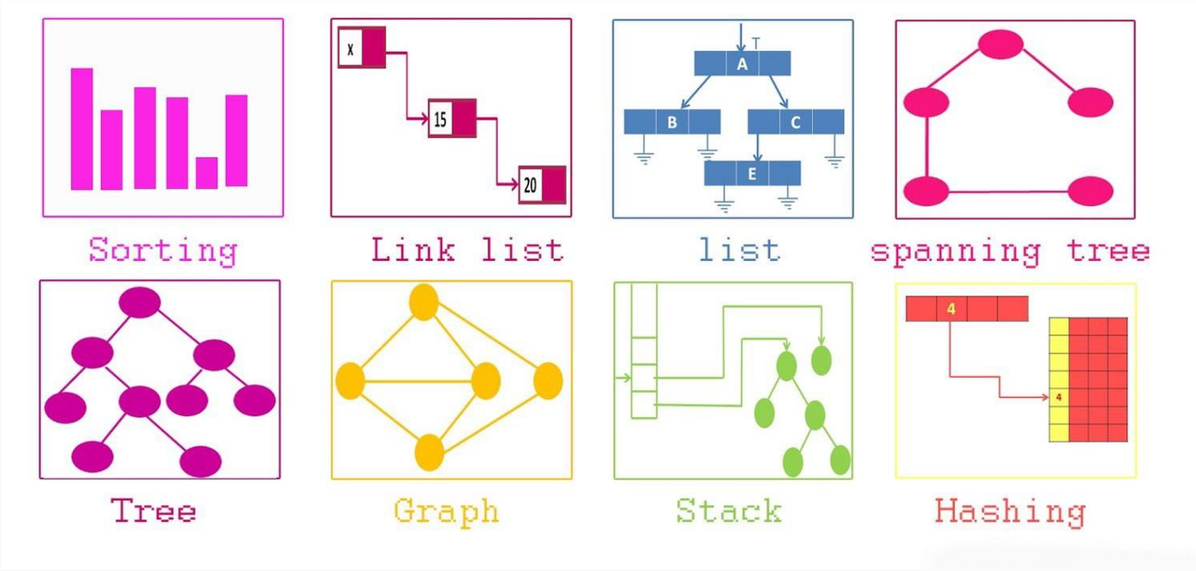
Arreglos

¿Qué es una estructura de datos?

Son colecciones que mantienen diferentes relaciones entre los datos que almacenan. Dichas estructuras permiten un eficiente acceso y modificación de dichos datos

Cada estructura de datos tiene un uso para diferentes problemas. Por ejemplo, una estructura de datos puede utilizarse para guardar información de personas en base a sus nombres

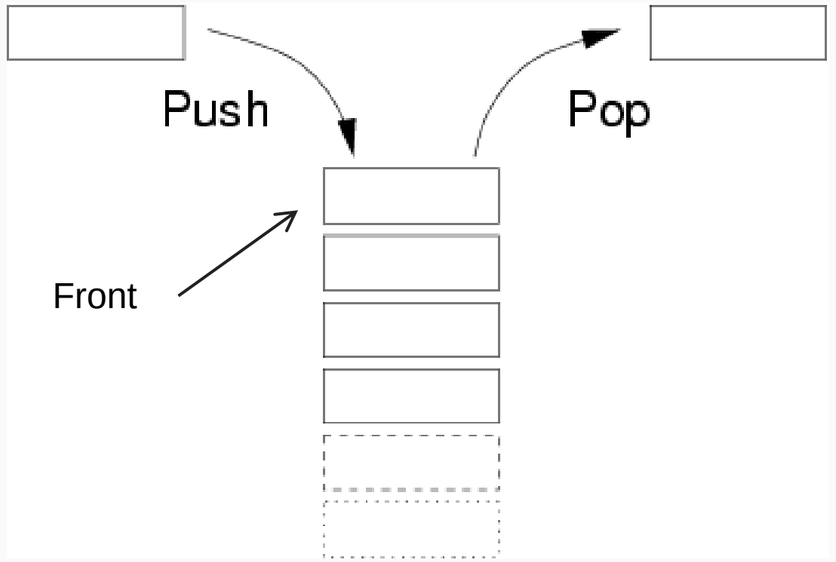
Son estructuras programadas para almacenar datos en memoria (RAM o disco) para que varias operaciones puedan realizarse de manera fácil



Stack

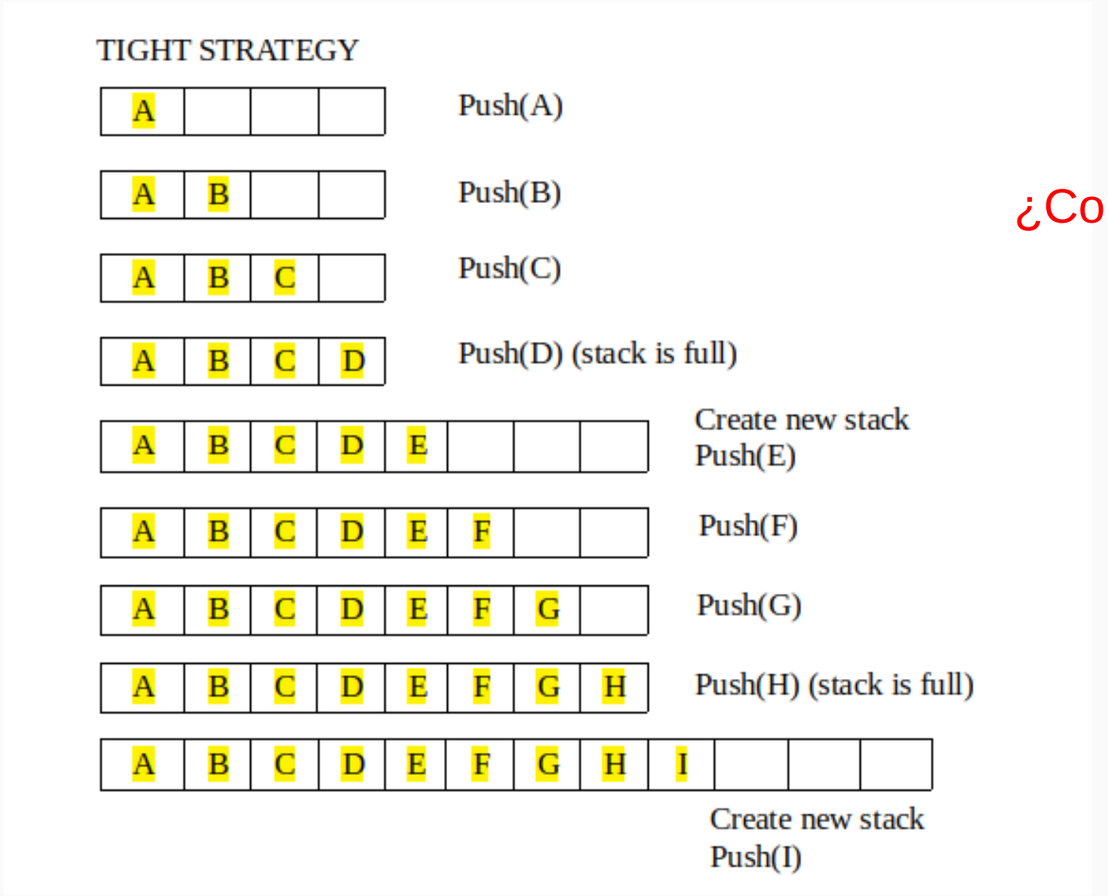
Es una estructura de datos básica que se puede representar como una pila.

Su implementación se llama también LIFO (Last In First Out)



Stack with Arrays

```
Class Stack {
  Private:
      int* array;
      int capacity;
      int size;
  public:
        Stack();
        void push(int data);
        int pop();
        void display();   }
```

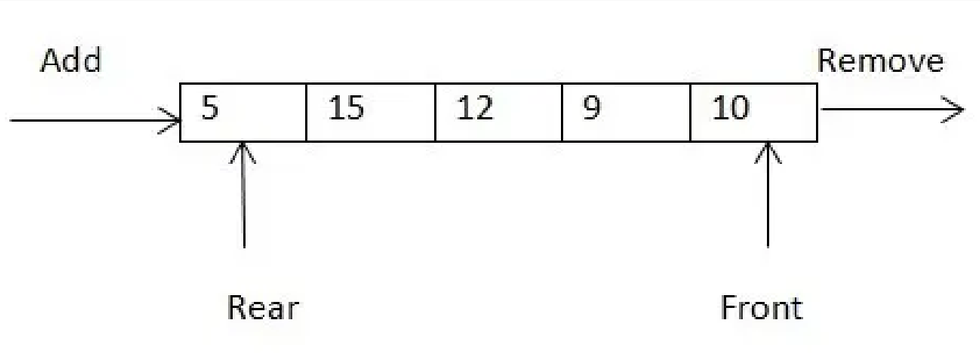


¿Complejidad Computacional?

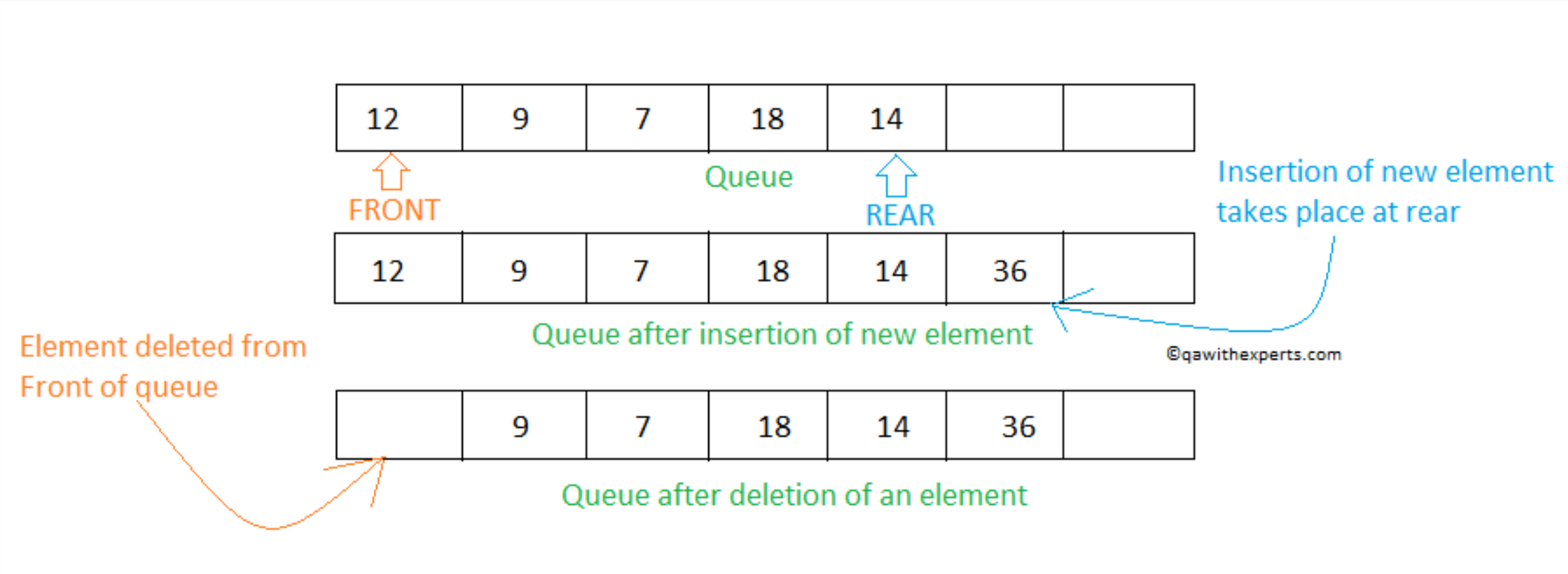
Queue

Es una estructura de datos básica muy similar a stack.

Su implementación se llama también FIFO (First In First Out)

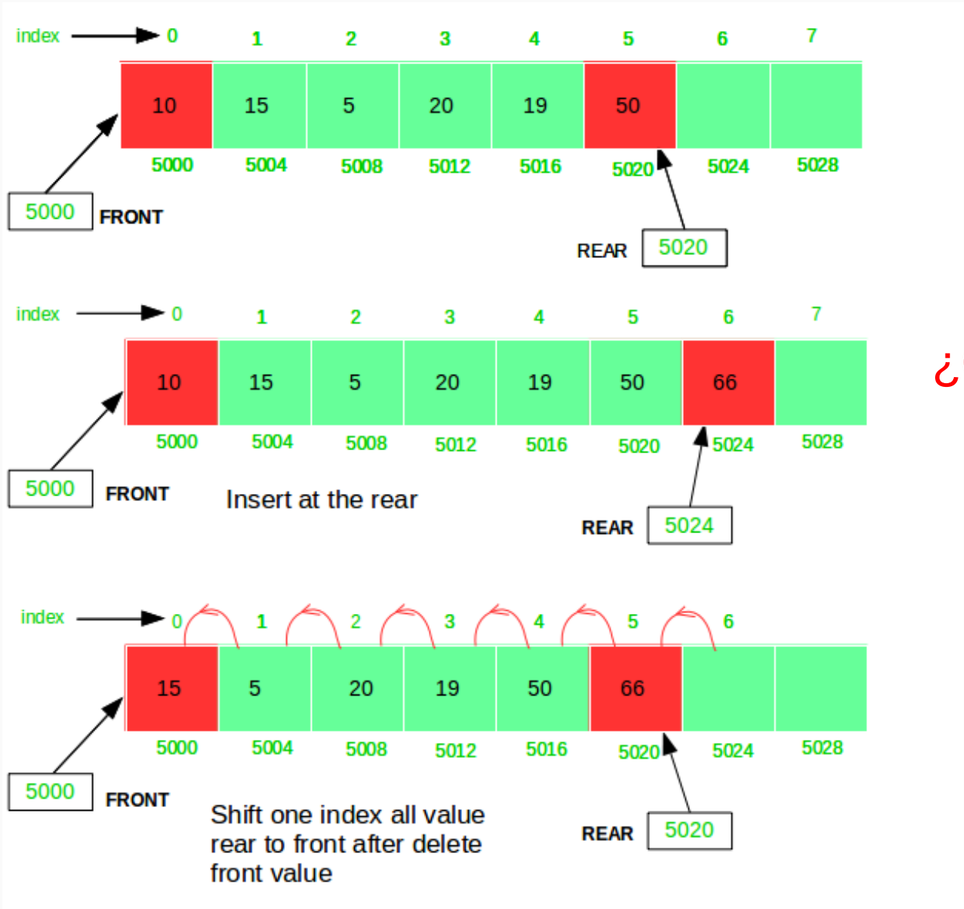


Queue with Arrays



Queue with Arrays

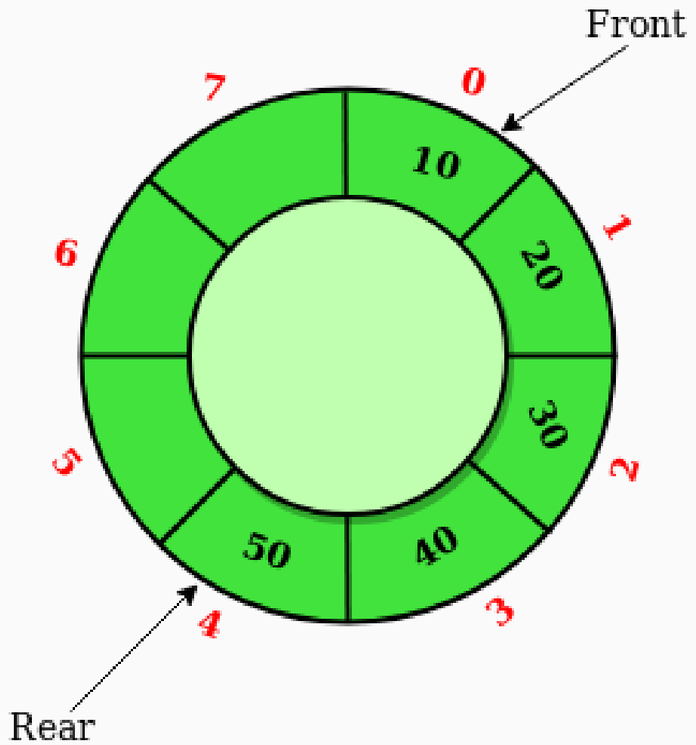
```
Class Queue {
  private:
      int* array;
      int capacity;
      int size;
  public:
        Queue();
        void enqueue(int data);
        int dequeue();
        void display();   }
```



¿Complejidad Computacional?

Queue with Circular Arrays

```
Class Queue {
  private:
      int* array;
      int capacity;
      int front, rear;
  public:
        Queue();
        void enqueue(int data);
        int dequeue();
        void display();   }
```

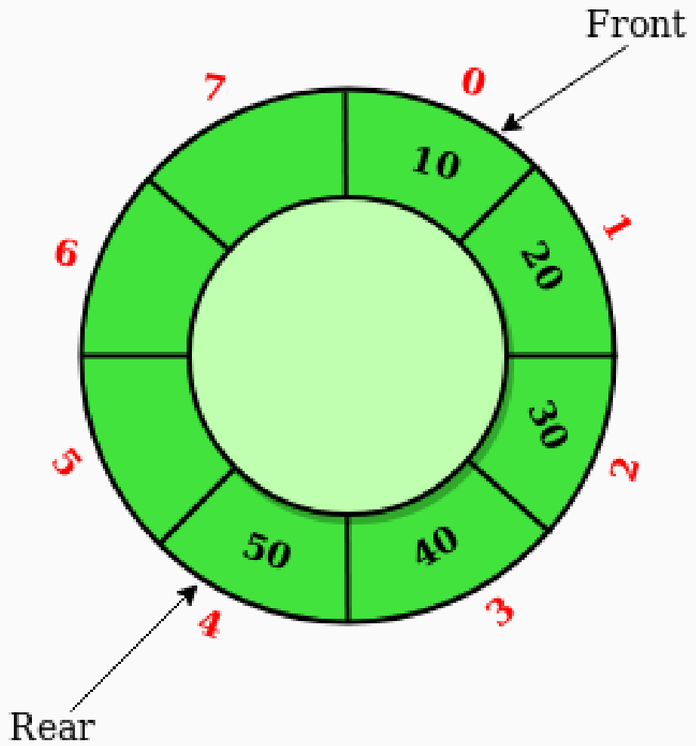


Condiciones para:

- Cola vacía ?
- Cola llena ?

Queue with Circular Arrays

```
Class Queue {
  private:
      int* array;
      int capacity;
      int front, rear;
  public:
        Queue();
        void enqueue(int data);
        int dequeue();
        void display();   }
```



Condiciones para:

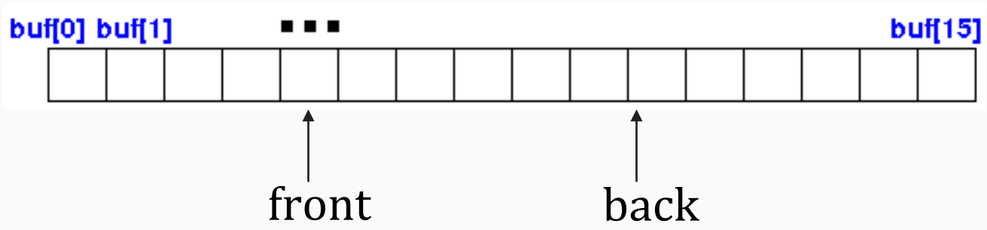
- Cola vacía ?
- Cola llena ?

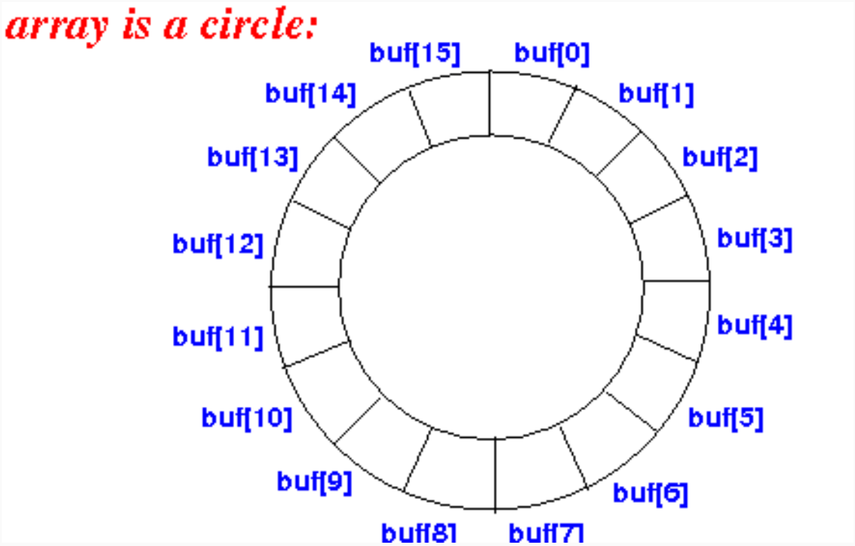
¿Complejidad Computacional?

Insertar: O(1)
Eliminar: O(1)
Problema: redimensionar

Circular Array

- Operaciones:
  - pop_front()
  - pop_back()
  - push_front(data)
  - push_back(data)
  - **insert(data, position)**
  - Otras:
    - is_empty(), is_full()
- Complejidad:
  -
  -
  -
  -
  - O(n)





Circular Array (Homework)

T front(); *// Retorna el elemento al comienzo*

T back(); *// Retorna el elemento al final*

void push_front(T); *// Agrega un elemento al comienzo*

void push_back(T); *// Agrega un elemento al final*

T pop_front(); *// Remueve el elemento al comienzo*

T pop_back(); *// Remueve el elemento al final*

Void insert(T, int); *// Inserta el elemento en la posición indicada*

T operator[](int); *// Retorna el elemento en la posición indicada*

bool empty(); *// Retorna si el array está vacía o no*

bool full(); *// Retorna si el array está lleno o no*

int size(); *// Retorna el tamaño del array*

void clear(); *// Elimina todos los elementos del array*

void sort(); *// Ordena el array* ***(use el algoritmo eficiente que le tocó en la práctica de ordenamiento****)*

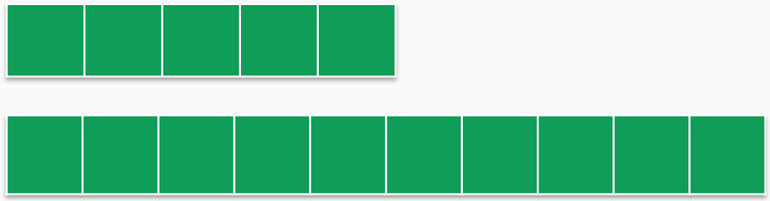
T* reverse(); // Devuelve una copia d*el array revertido*

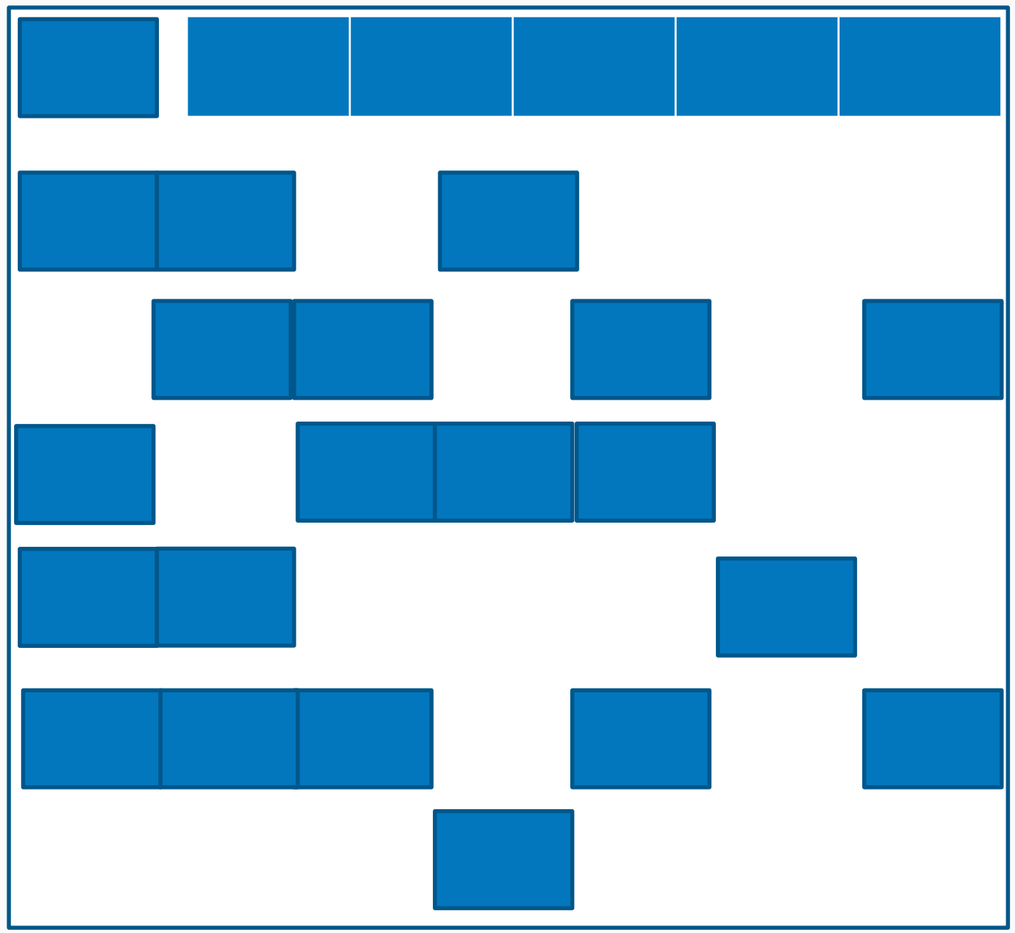
Array

- Problema de asignación de memoria

int *array = new int[5]

Redimensionar?





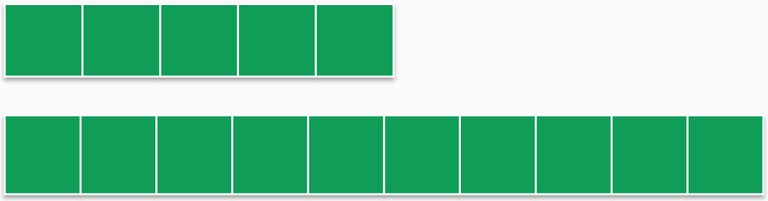
RAM

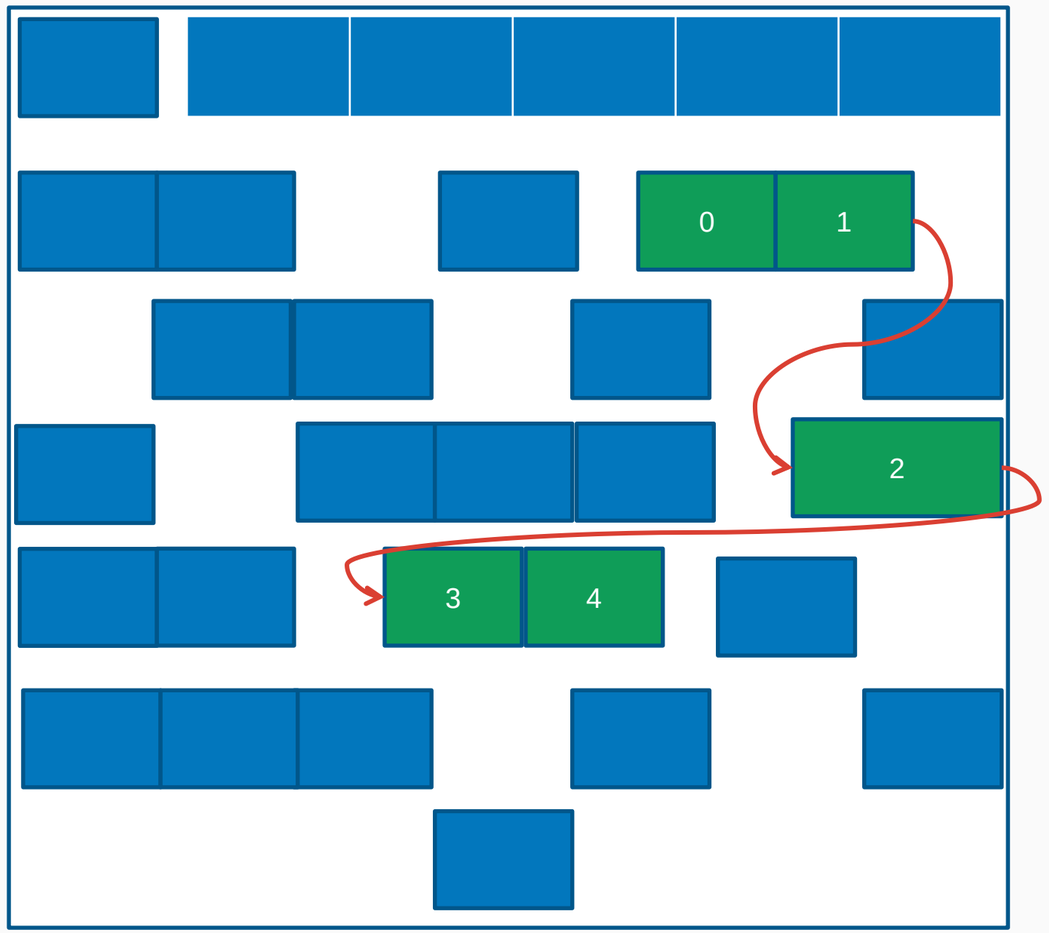
Array

- Problema de asignación de memoria

int *array = new int[5]

Redimensionar?





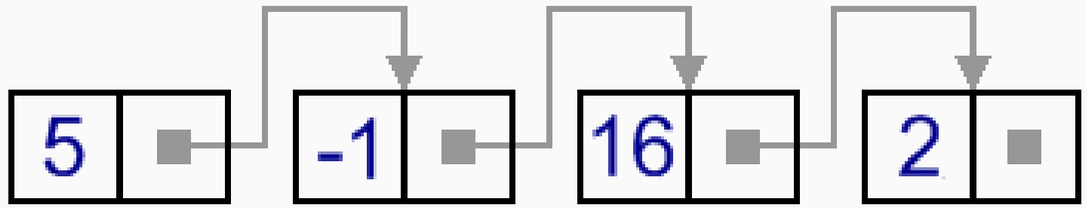
RAM

Solucion: Lista enlazada

Listas Enlazadas

Forward List (Simple Linked List)

Es un contenedor secuencial, el cual permite tener datos en espacios de almacenamiento no relacionados (memoria)

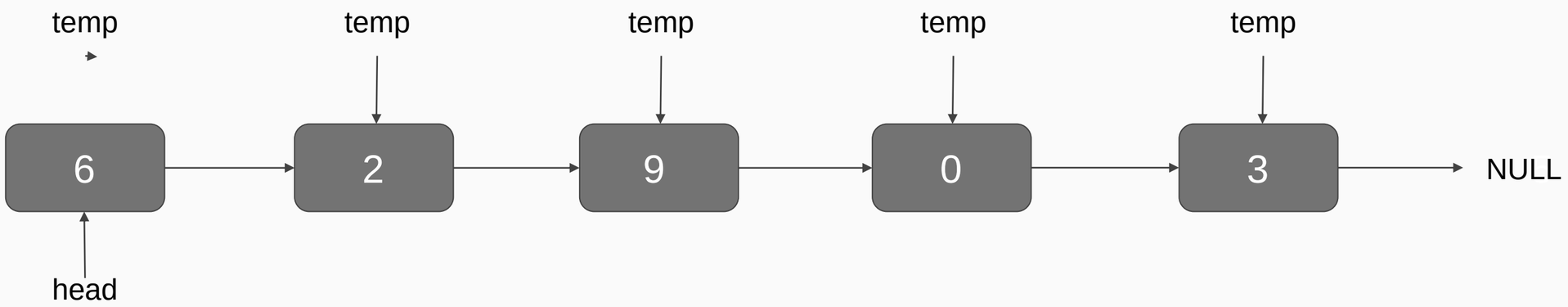


Forward List (Node)

```
struct Node {
    int data;
    Node* next;
};
```

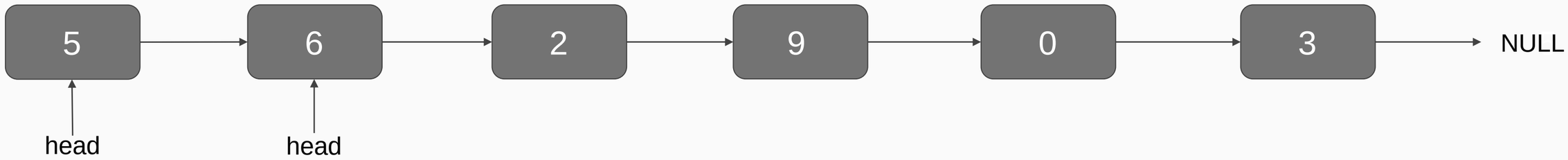
```
class List {
    private:
    Node* head;
     …
};
```

Forward List




```
Nodo* temp = head;
while (temp != NULL) {
    cout<< temp->data;
    temp =  temp->next;
}
```

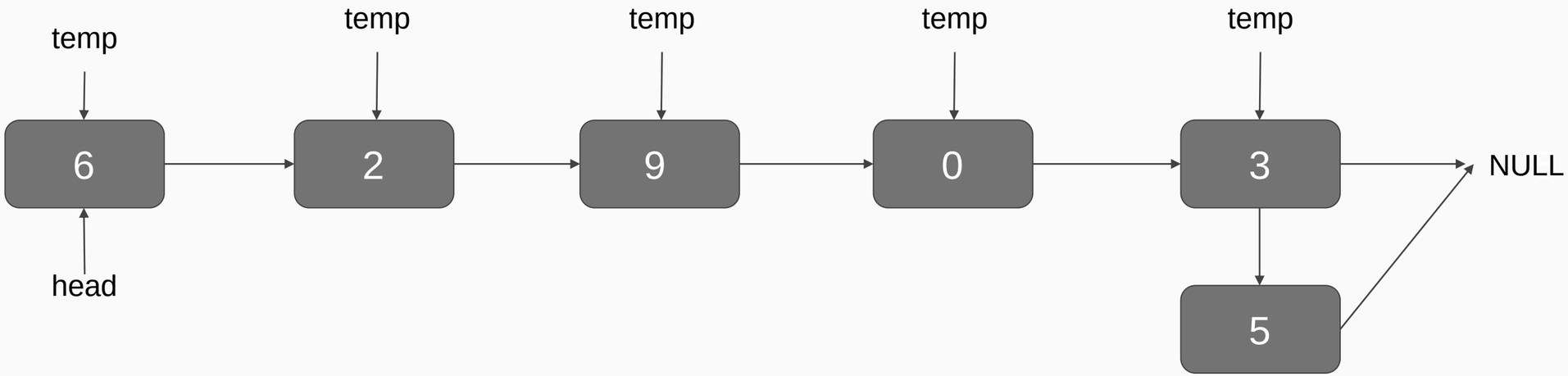
Forward List (push front 5)




```
Nodo* nodo = new Nodo;
nodo->data = 5;
nodo->next = head;
head = nodo;
```

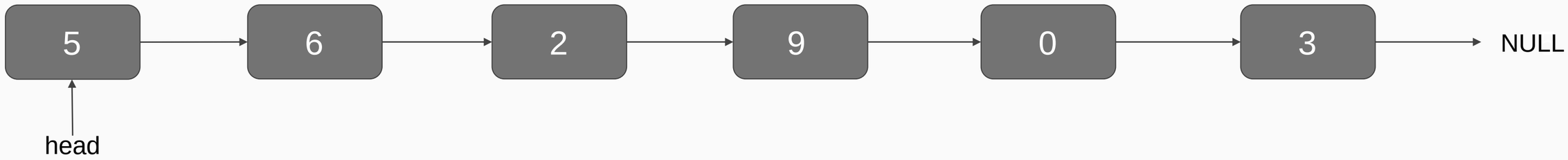
¿Y si la lista esta vacía?

Forward List (push back 5)



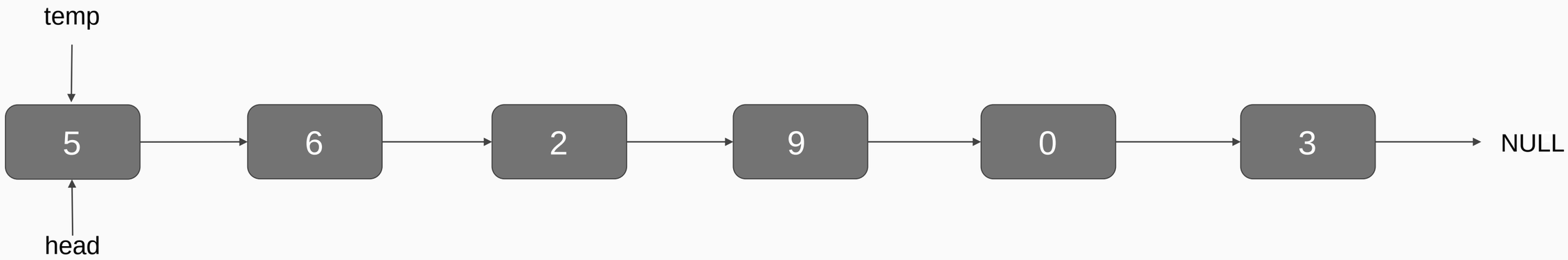

```
Nodo* nodo = new Nodo;
nodo->data = 5;
Nodo* temp = head;
while (temp->next !=NULL)
    temp = temp->next;
temp->next = nodo;
nodo->next = NULL;
```

Forward List (pop front)




```
Nodo* temp = head;
head = head->next;
delete temp;
```

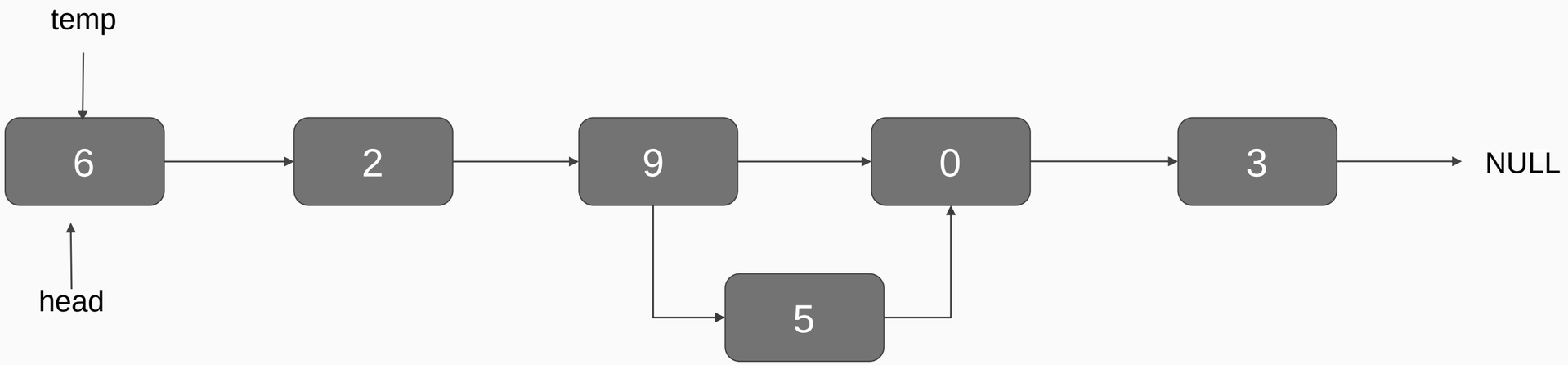
Forward List (pop back)




```
if(head->next == NULL)
{
    delete head;
    head = NULL;
}
```

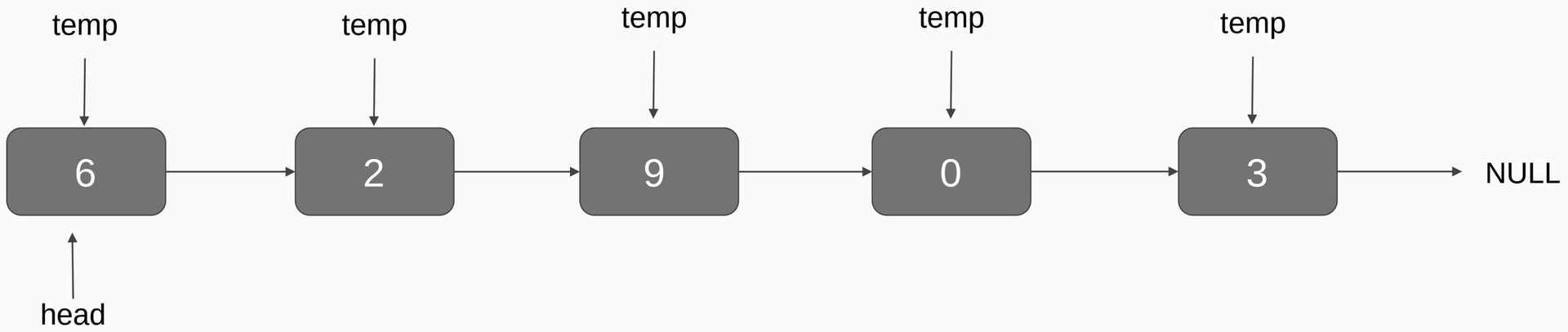
```
else
{
    Nodo* temp = head;
    while(temp->next->next != NULL)
        temp = temp->next;
    delete temp->next;
    temp->next = NULL;
}
```

Forward List (insert 5 at location 3)




```
Nodo* nodo = new Nodo(5);
Nodo* temp = head;
int i = 0;
while(i++ < pos - 1) temp = temp->next;
nodo->next = temp->next;
temp->next = nodo;
```

Forward List (clear)

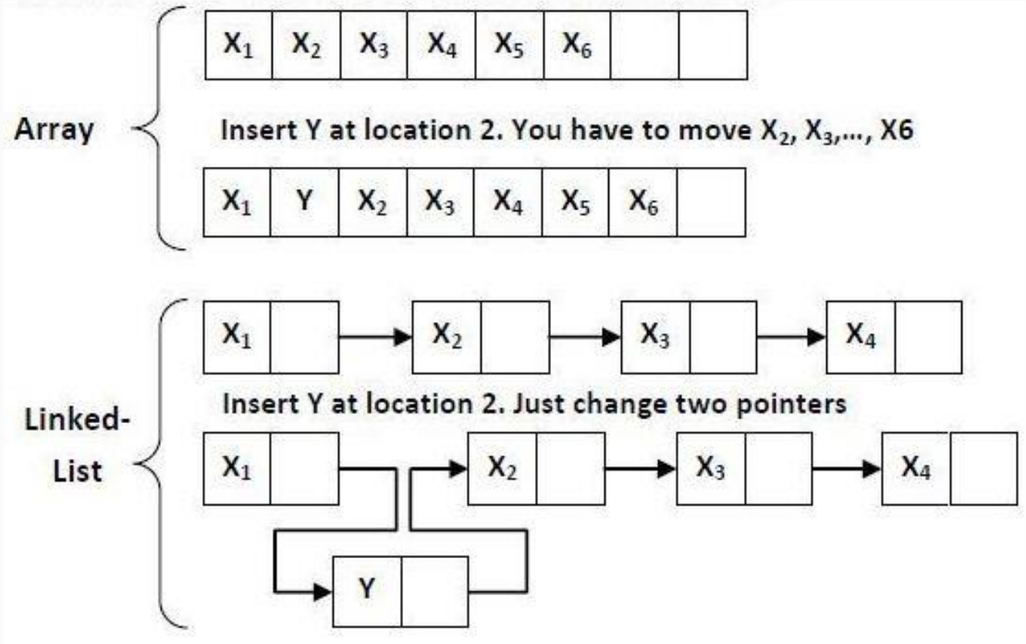



```
while(head != NULL)
{
    Nodo* temp = head;
    head = head->next;
    delete temp;
}
head = NULL;
```

Forward List VS Array

¿Cuál es la diferencia con arreglo?

- Ubicación en la memória
- Tiempos de acceso
- Tamaño
- Dimensiones



Forward List

1. ¿Cuánto tiempo demora encontrar un elemento al comienzo? ¿Al final? ¿En cualquier posición?

O(1), O(n) y O(n)

2. ¿Y para insertar un elemento después del primer nodo? ¿Después del último nodo? ¿Después de cualquier nodo?

O(1), O(n) y O(n)

3. ¿Cómo sería el caso 2 pero antes del nodo?

O(1), O(n) y O(n)

Forward List

1. ¿Cuánto tiempo demora borrar un elemento al comienzo? ¿Al final? ¿En cualquier posición?

O(1), O(n) y O(n)

2. ¿Y para obtener el siguiente elemento de un nodo al comienzo? ¿En cualquier posición?

O(1) y O(n)

3. ¿Cómo sería para obtener el nodo anterior al final? ¿En cualquier posición?

O(n) y O(n)

Forward List (Homework)

T front(); *// Retorna el elemento al comienzo*

T back(); *// Retorna el elemento al final*

void push_front(T); *// Agrega un elemento al comienzo*

void push_back(T); *// Agrega un elemento al final*

T pop_front(); *// Remueve el elemento al comienzo*

T pop_back(); *// Remueve el elemento al final*

T operator[ ](int); *// Retorna el elemento en la posición indicada*

bool empty(); *// Retorna si la lista está vacía o no*

int size(); *// Retorna el tamaño de la lista*

void clear(); *// Elimina todos los elementos de la lista*

void sort(); *// Implemente un algoritmo de ordenacion con listas enlazadas)*

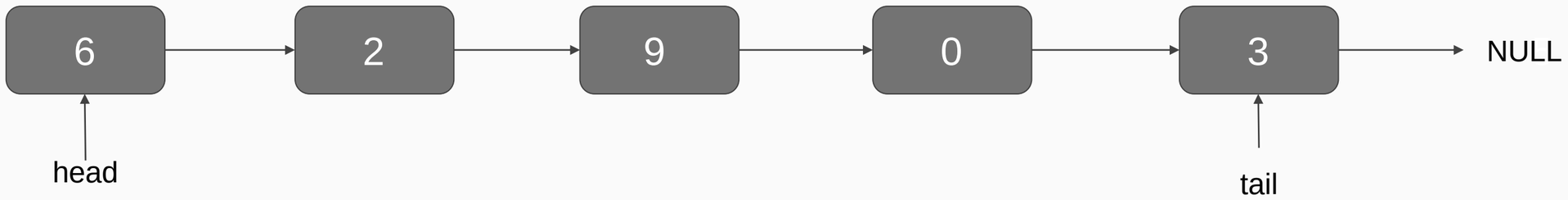
void reverse(); *// Revierte la lista*

Forward List (Ejercicios)

1. Revertir una lista enlazada
2. **Verificar si la lista es palíndromo o no.**
3. **Mezclar dos listas ordenadas en una nueva lista también ordenada.**
4. Implementar el algoritmo de Insertion Sort con listas enlazadas.

Lists (improves)

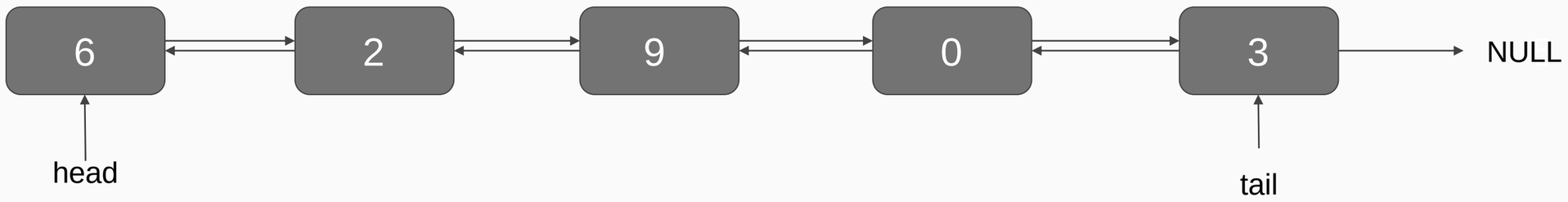
¿Cuánto demoraría concatenar dos listas?



¿Cómo borraríamos el último elemento?

¿Cómo imprimiríamos la lista al revés?

Doubly Linked List



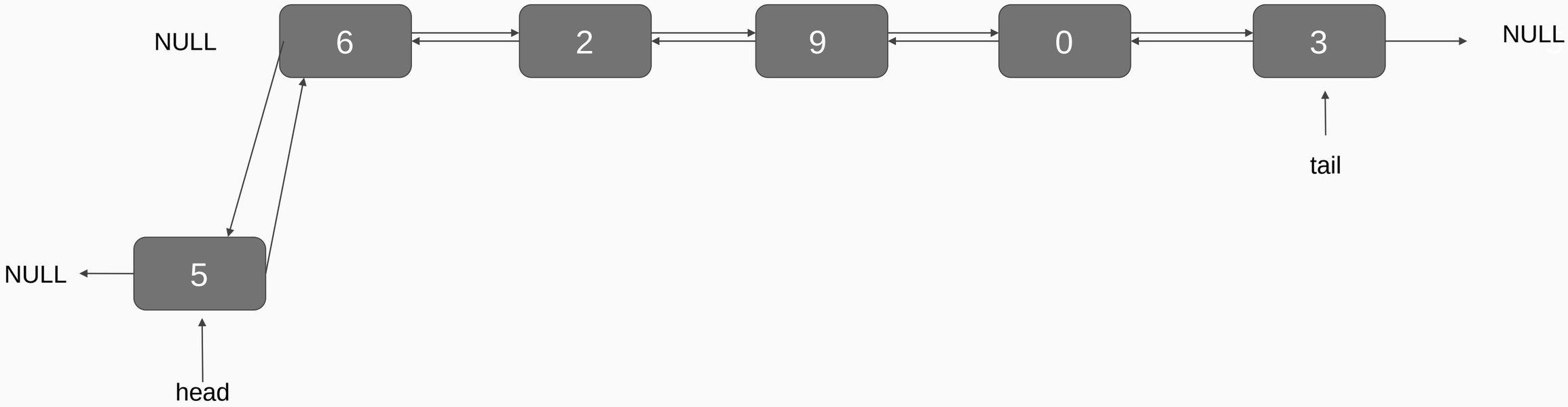
¿Cuáles son las ventajas?

Doubly Linked List (node)

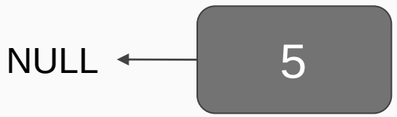
```
struct Node {
        int data;
        Node* next;
        Node* prev;
};
```

```
class List {
    private:
            Node* head;
            Node* tail;
};
```

Doubly Linked List (push front 5)

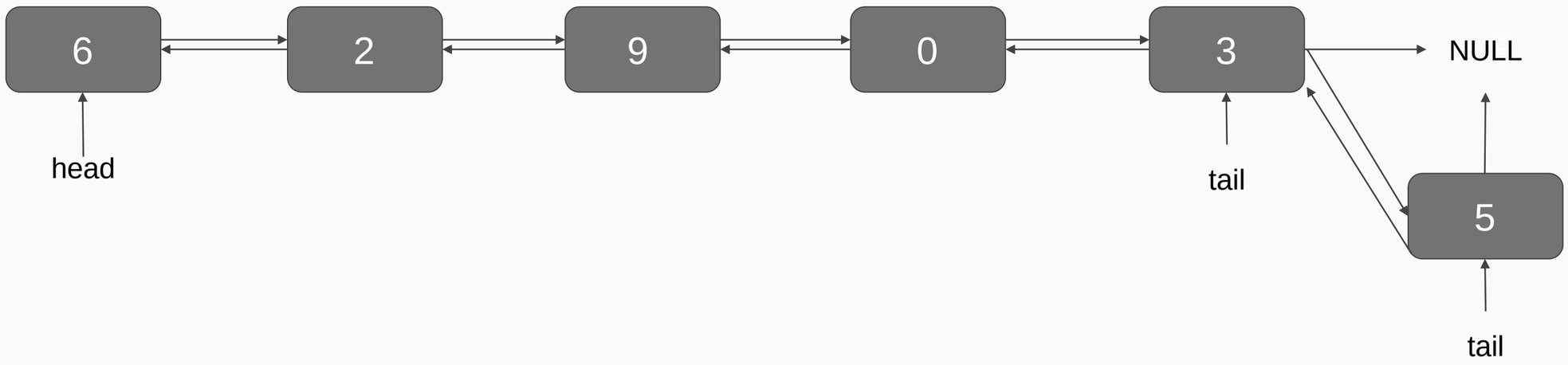


Doubly Linked List (push front 5)

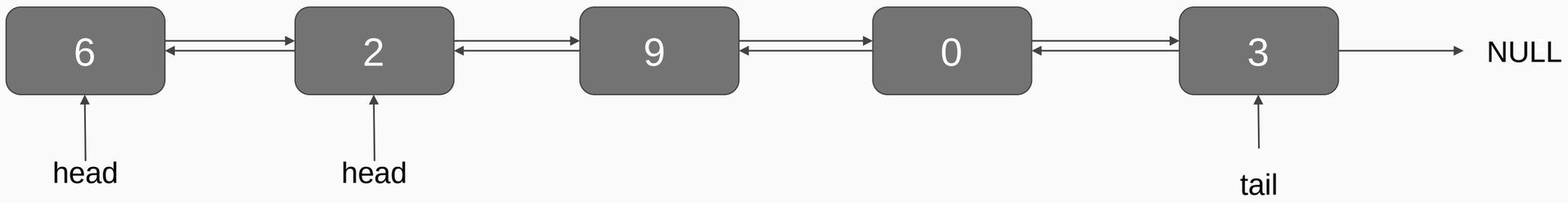


1- Crear nodo con el dato
   Node* nodo = Node(5)
   nodo->prev = nullptr;
2- Nuevo nodo apunta a la cabeza
   nodo ->next = head
4- Cabeza actual apunta a nuevo nodo
   head -> prev = nodo
5- Mover la cabeza al nuevo nodo
   head = nodo

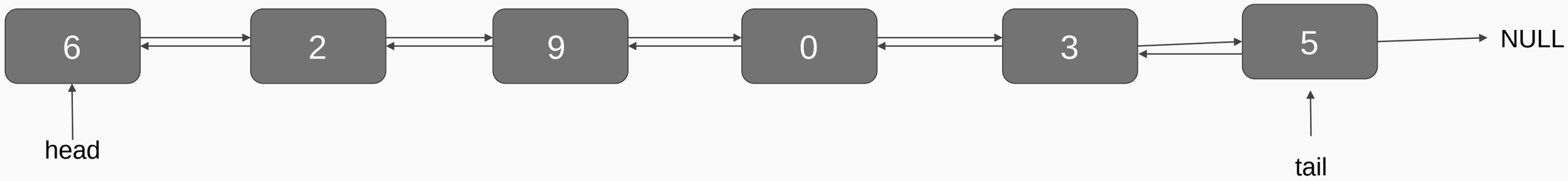
Doubly Linked List (push back 5)



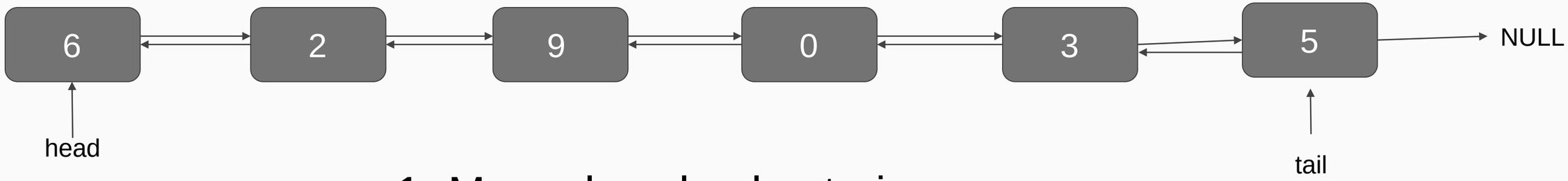
Doubly Linked List (pop front)



Doubly Linked List (pop back)

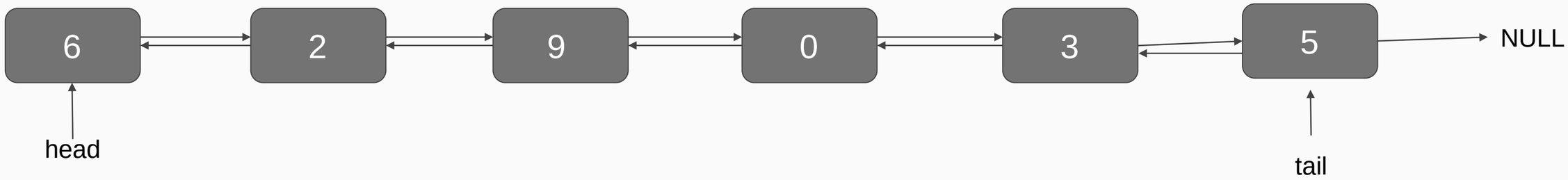


Doubly Linked List (pop back)



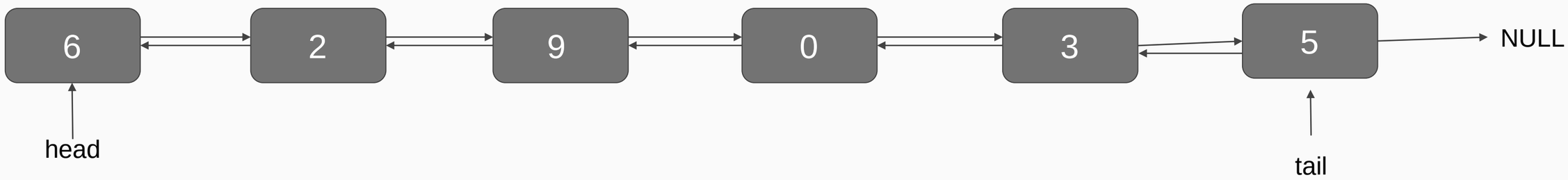
1- Mover la cola al anterior
tail = tail -> prev
2- Liberar cola
delete tail->next;
3- Cola apunta a null
tail->next = nullptr;

Doubly Linked List (pop front)



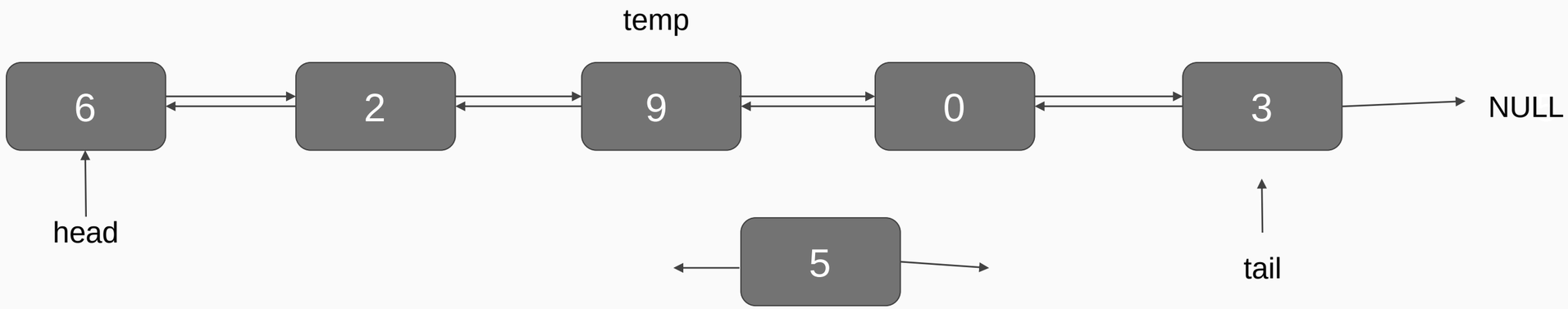
6

Doubly Linked List (pop front)

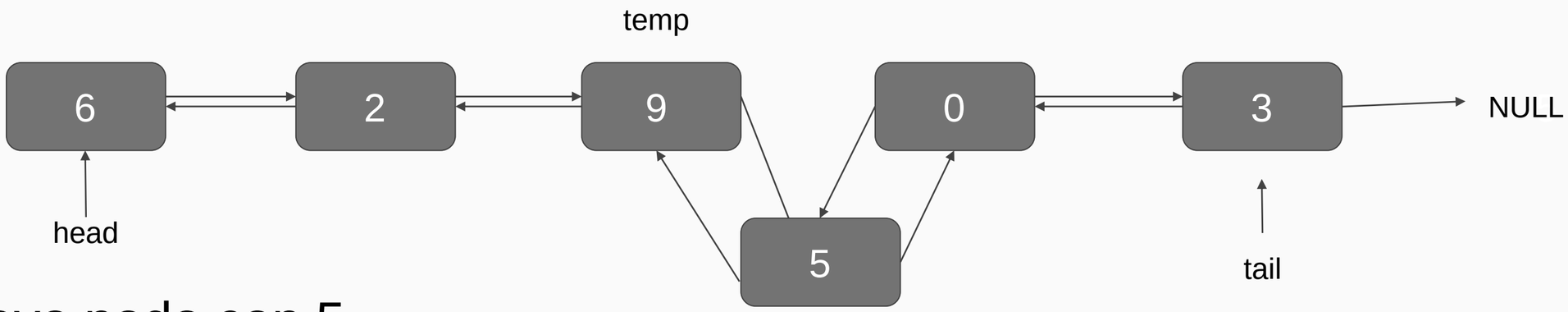


1- Moviendo la cabeza al siguiente nodo
   head = head -> next
2- Borrar la anterior cabeza
   delete[] head->prev
3- Actualizamos punteror prev de la cabeza
   head -> prev = null

Doubly Linked List (insert 5 at 3)



Doubly Linked List (insert 5 at 3)



1- Crear nuevo nodo con 5
   Node* nodo = new Node(5)
2- Recorrer desde la cabeza
   Node* temp = head;
   i = 0; while(i++ < pos - 1) temp = temp->next;
3- Actualizar punteros del nuevo
   nodo->next = temp->next;
   nodo->prev = temp

4- Actualizar punteros de temp
   temp->next->prev = nodo
   temp->next = nodo

Doubly Linked List

1. ¿Cuánto tiempo demora encontrar un elemento al comienzo? ¿Al final? ¿En cualquier posición?

O(1), O(1) y O(n)

2. ¿Y para insertar un elemento después del primer nodo? ¿Después del último nodo? ¿Después de cualquier nodo?

O(1), O(1) y O(n)

3. ¿Cómo sería el caso 2 pero antes del nodo?

O(1), O(1) y O(n)

Doubly Linked List (Homework)

T front(); *// Retorna el elemento al comienzo*

T back(); *// Retorna el elemento al final*

void push_front(T); *// Agrega un elemento al comienzo*

void push_back(T); *// Agrega un elemento al final*

T pop_front(); *// Remueve el elemento al comienzo*

T pop_back(); *// Remueve el elemento al final*

void insert(T, int); *// Inserta en cualquier posición*

void remove(int); *// Remueve en cualquier posicion*

T operator[ ](int); *// Retorna el elemento en la posición indicada*

bool empty(); *// Retorna si la lista está vacía o no*

int size(); *// Retorna el tamaño de la lista*

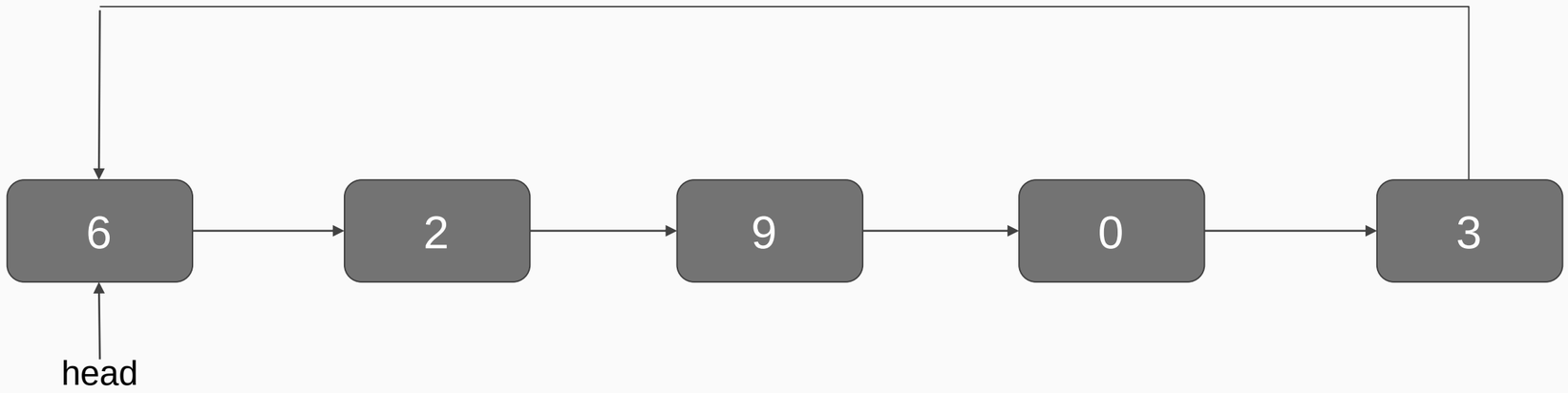
void clear(); *// Elimina todos los elementos de la lista*

void reverse(); *// Revierte la lista*

Doubly Linked List (Ejercicios)

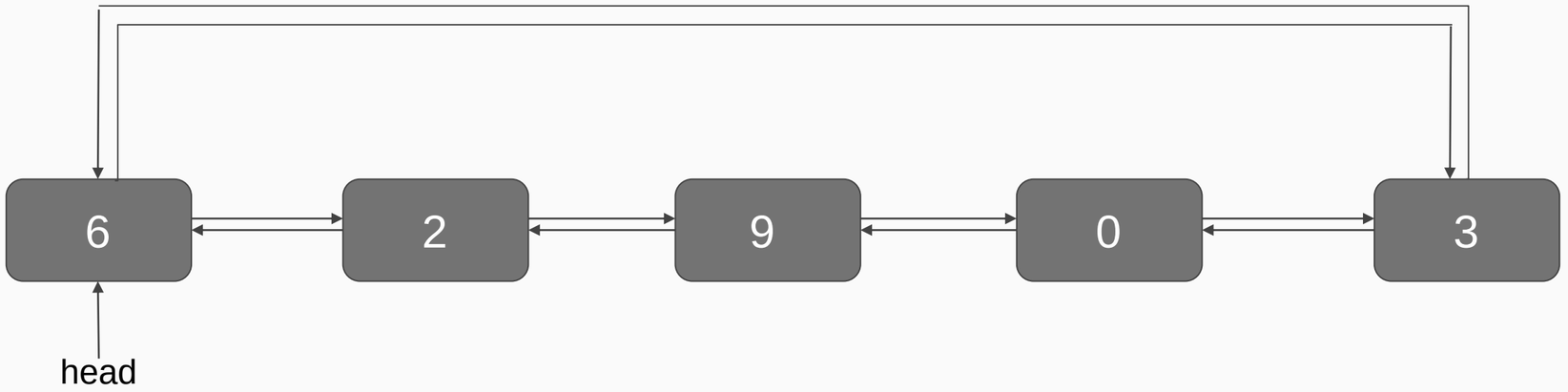
1. **Verificar si la lista es palíndromo o no.**
2. **Implementar la función SortedInsert el cual inserta un elemento en la lista manteniendo un orden ascendente.**
3. **Aplicar el operador unión de conjuntos en dos listas**
4. **Eliminar elementos repetidos de una lista.**

Circular Linked List



¿Qué usos le darían a una lista circular simplemente enlazada?

Circular Doubly Linked List

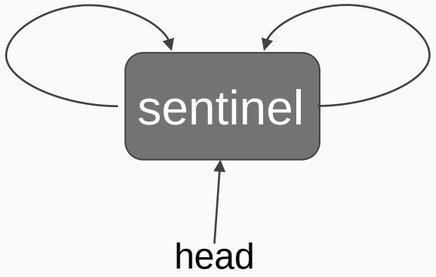


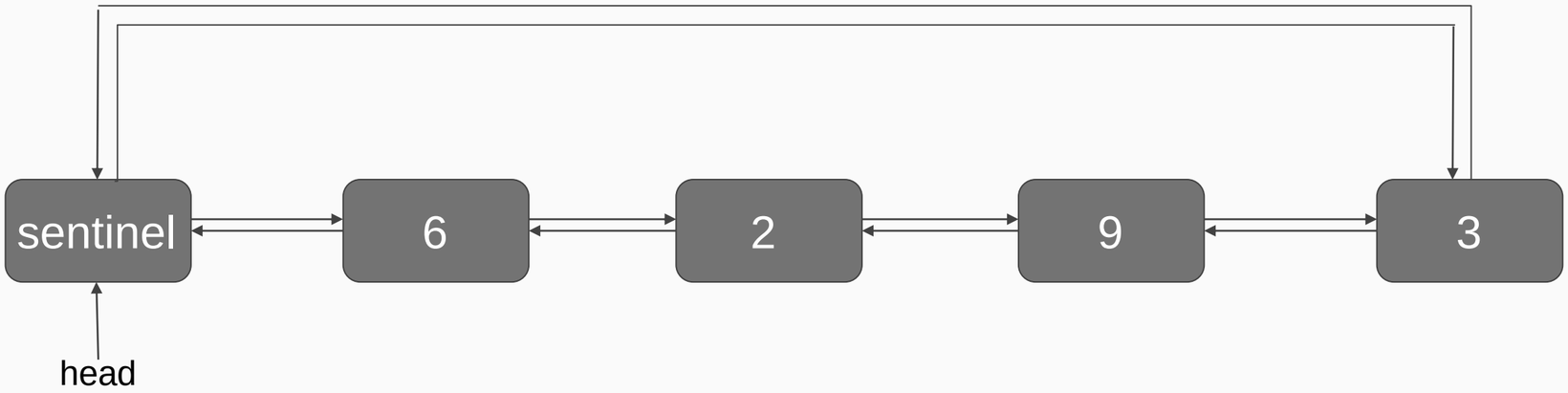
Siempre habrá un next y un prev

¿Qué usos le darían a una lista circular doblemente enlazada?

Circular Doubly Linked List

Nodo Centinela



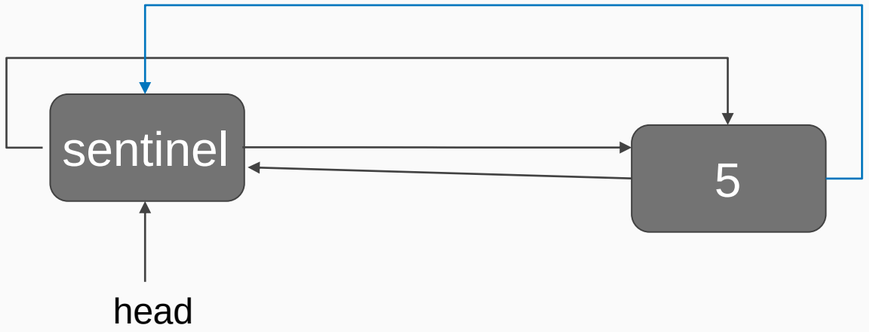


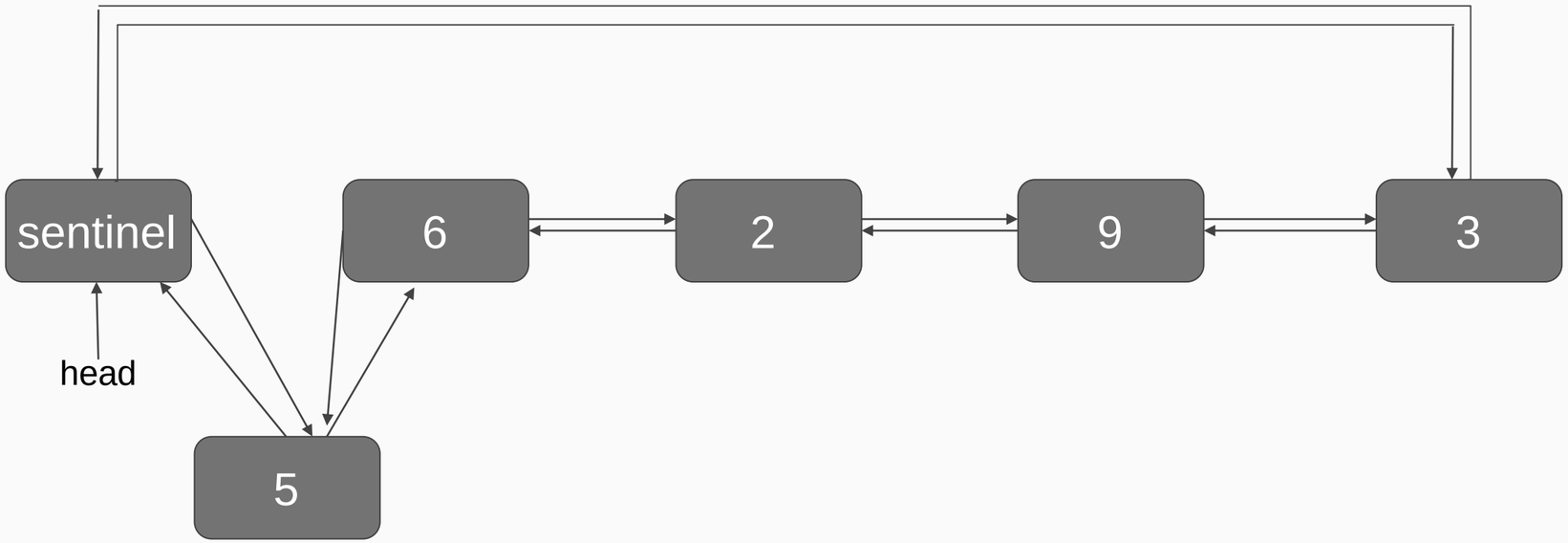
No existen casos especiales, cada nodo siempre tiene uno anterior y uno siguiente.

https://www.programmersought.com/article/76696946762/

- Constructor de la Lista

head = new Node()
head->next = head
head->prev = head




sentinel
6
2
9
3
head
5

Circular Doubly List (Homework)

T front(); *// Retorna el elemento al comienzo*

T back(); *// Retorna el elemento al final*

void push_front(T); *// Agrega un elemento al comienzo*

void push_back(T); *// Agrega un elemento al final*

T pop_front(); *// Remueve el elemento al comienzo*

T pop_back(); *// Remueve el elemento al final*

void insert(T, int); *// Inserta en cualquier posición*

void remove(int); *// Remueve en cualquier posicion*

T operator[ ](int); *// Retorna el elemento en la posición indicada*

bool empty(); *// Retorna si la lista está vacía o no*

int size(); *// Retorna el tamaño de la lista*

void clear(); *// Elimina todos los elementos de la lista*

void reverse(); *// Revierte la lista*

Lists with templates

```
template <typename T>
struct Node {
    T data;
    Node<T>* next;
    Node<T>* prev;
};
```

```
List<int>* test = new List<int>()
List<char>* test = new List<char>()
List<float>* test = new List<float>()
```

```
typename <typename T>
class List {
  private:
    Node<T>* head;
    Node<T>* tail;
};
```

Recuerden...

El curso se enfoca en que entiendan cada estructura de datos, sus algoritmos y usos comunes.

No hay que reinventar la rueda, en el mercado lo más probable es que utilicen otras implementaciones, por ejemplo de la Standard Template Library (STL).

De todas formas, habrá situaciones en las que probablemente implementarán sus propias estructuras.

UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Welcome to
Algorithms and Data Structures!
CS2100