



Listas Encadeadas

PrepTech Google

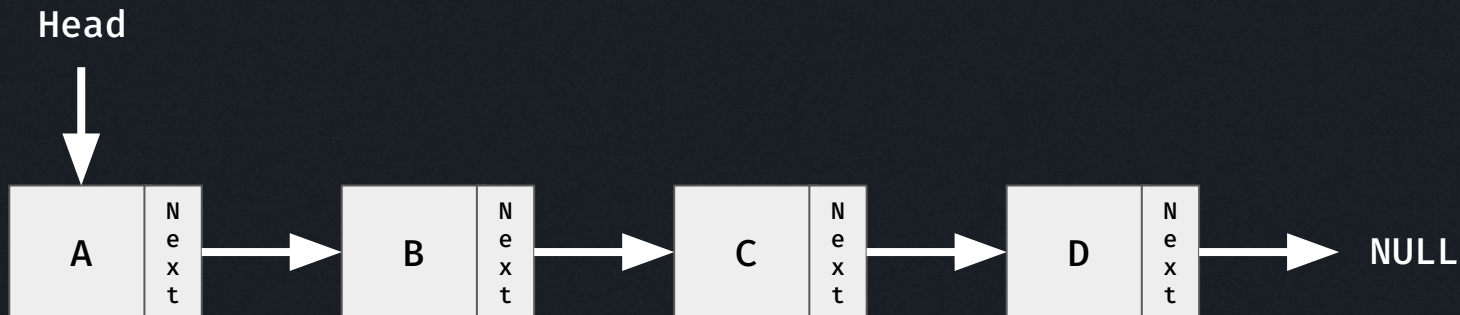
Problema

Playlist de músicas:

- Sem limite de músicas que podem ser adicionadas
- Músicas podem ser adicionadas em qualquer posição da playlist
- Usuários podem remover qualquer música da playlist
- Usuários podem trocar a ordem das músicas

Listas encadeadas

- Estrutura de dados linear
- Nós (nodes) da lista armazenam dados e uma referência (**next**) para o próximo nó
- A referência **next** do último nó da lista aponta para **null**
- O nó inicial da lista geralmente é chamado de **head**



Tipos

- **Singly linked list**
- Doubly linked list
- Circular linked list

Arrays x Listas encadeadas

Arrays

- Tamanho fixo: resizing é “caro”
- Inserções e remoções podem ser ineficientes (geralmente envolve “shift” nos elementos)
- Acesso aleatório (index)
- Memória previamente alocada para os elementos
- Acesso sequencial é rápido

Listas Encadeadas

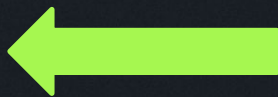
- Tamanho dinâmico
- Inserções e remoções são eficientes: não envolvem “shifts”
- Acesso sequencial
- Memória alocada dinamicamente quando necessário
- Acesso sequencial é mais lento

Operações

- Criação da lista
- Inserção
- Remoção
- Busca
- Inversão

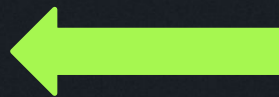
Criação

```
class Node {  
  constructor(data) {  
    this.data = data;  
    this.next = null;  
  }  
}
```



Nós da lista

```
class LinkedList {  
  constructor(){  
    this.head = null;  
  }  
}
```



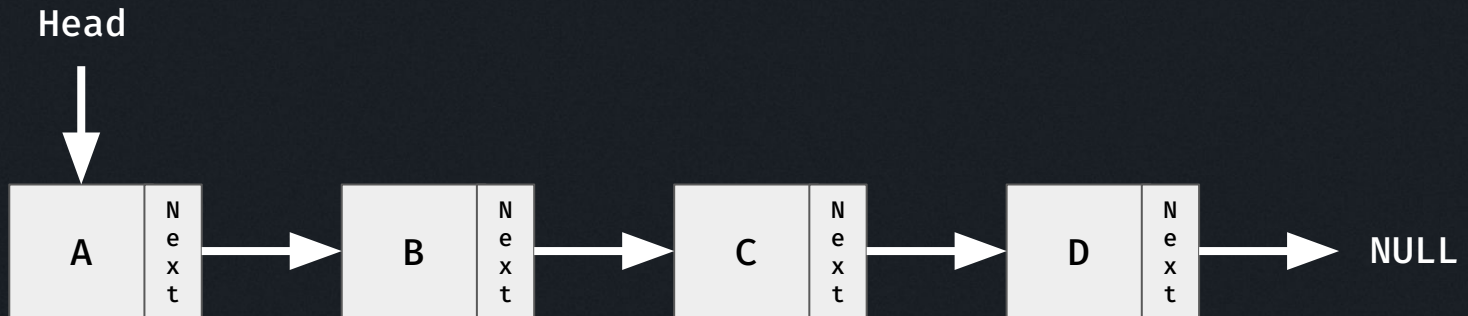
A lista encadeada é representada por uma referência para o primeiro nó

Inserção

- **3 Casos:**
 - Inserção no início
 - Inserção no fim
 - Inserção em uma posição específica

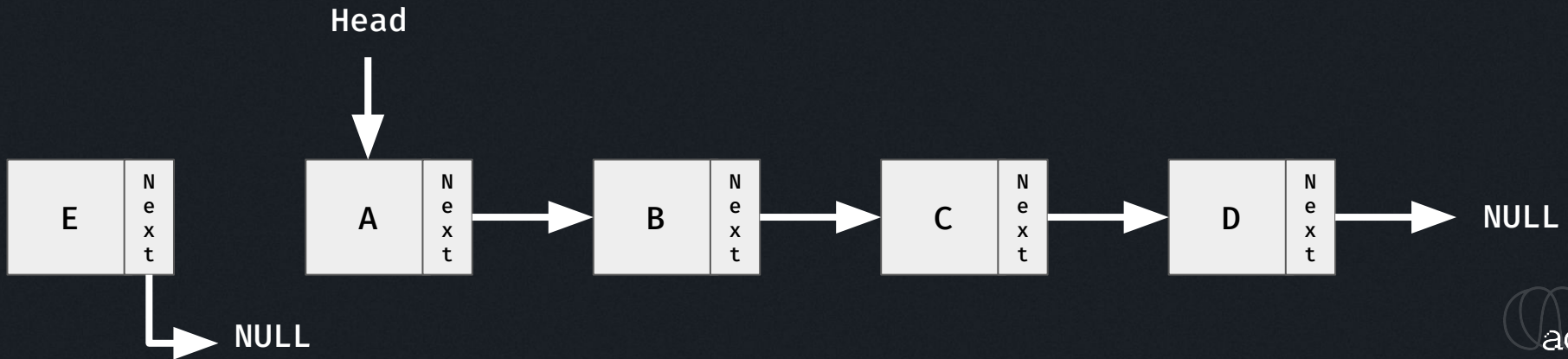
Inserção no início

- Criar um novo nó
- Atualiza a referência **next** do novo nó para apontar para **head**
- Atualiza a referência **head** da lista para apontar para o nó recém criado



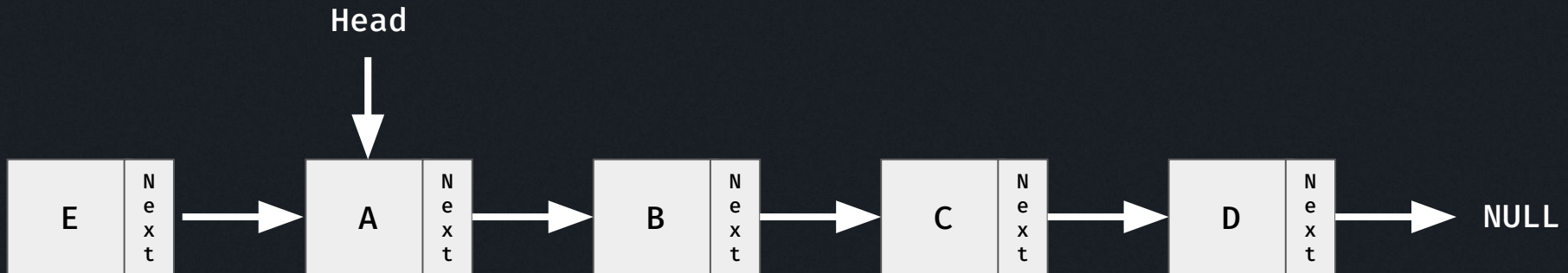
Inserção no início

- Criar um novo nó
- Atualiza a referência **next** do novo nó para apontar para **head**
- Atualiza a referência **head** da lista para apontar para o nó recém criado



Inserção no início

- Criar um novo nó
- Atualiza a referência **next** do novo nó para apontar para **head**
- Atualiza a referência **head** da lista para apontar para o nó recém criado



Inserção no início

- Criar um novo nó
- Atualiza a referência **next** do novo nó para apontar para **head**
- Atualiza a referência **head** da lista para apontar para o nó recém criado



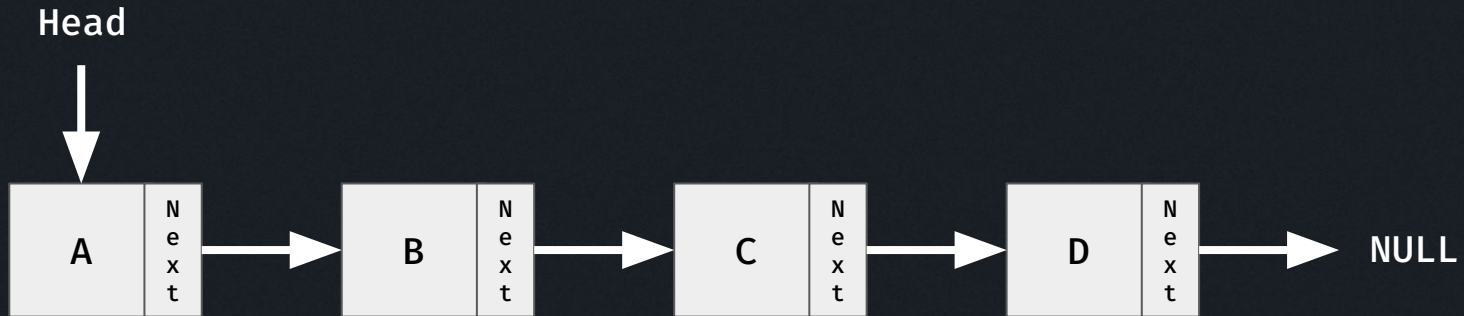
Inserção no início

- Criar um novo nó
- Atualiza a referência **next** do novo nó para apontar para **head**
- Atualiza a referência **head** da lista para apontar para o nó recém criado

```
insertBegin(data) {  
    const newNode = new Node(data);  
    newNode.next = this.head;  
    this.head = newNode;  
}
```

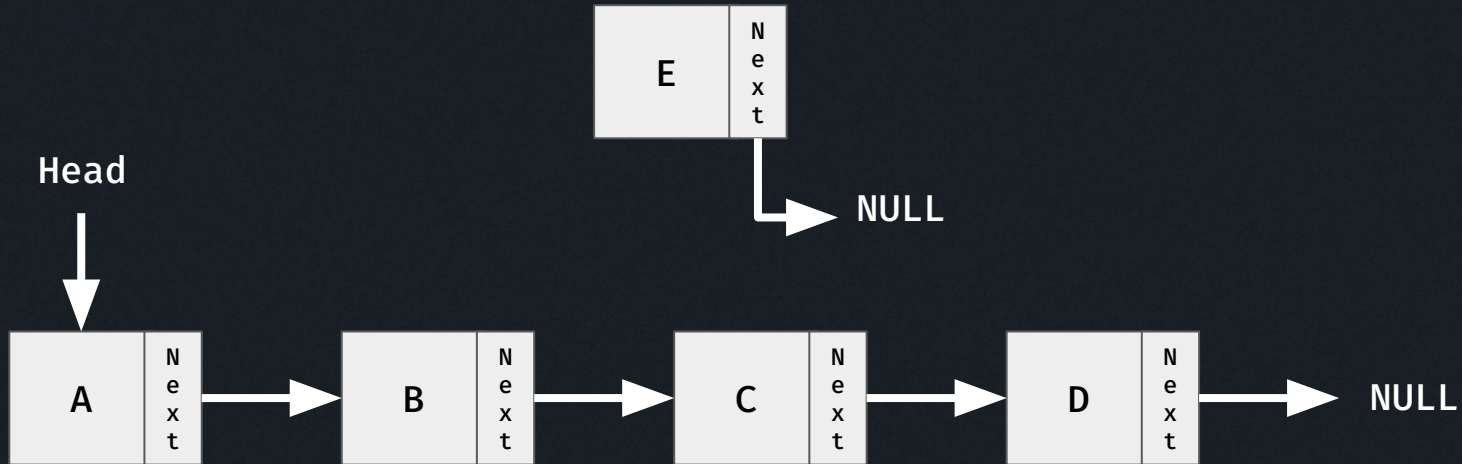
Inserção no fim

- Criar um novo nó
- Atualiza o último nó da lista apontar para o novo nó



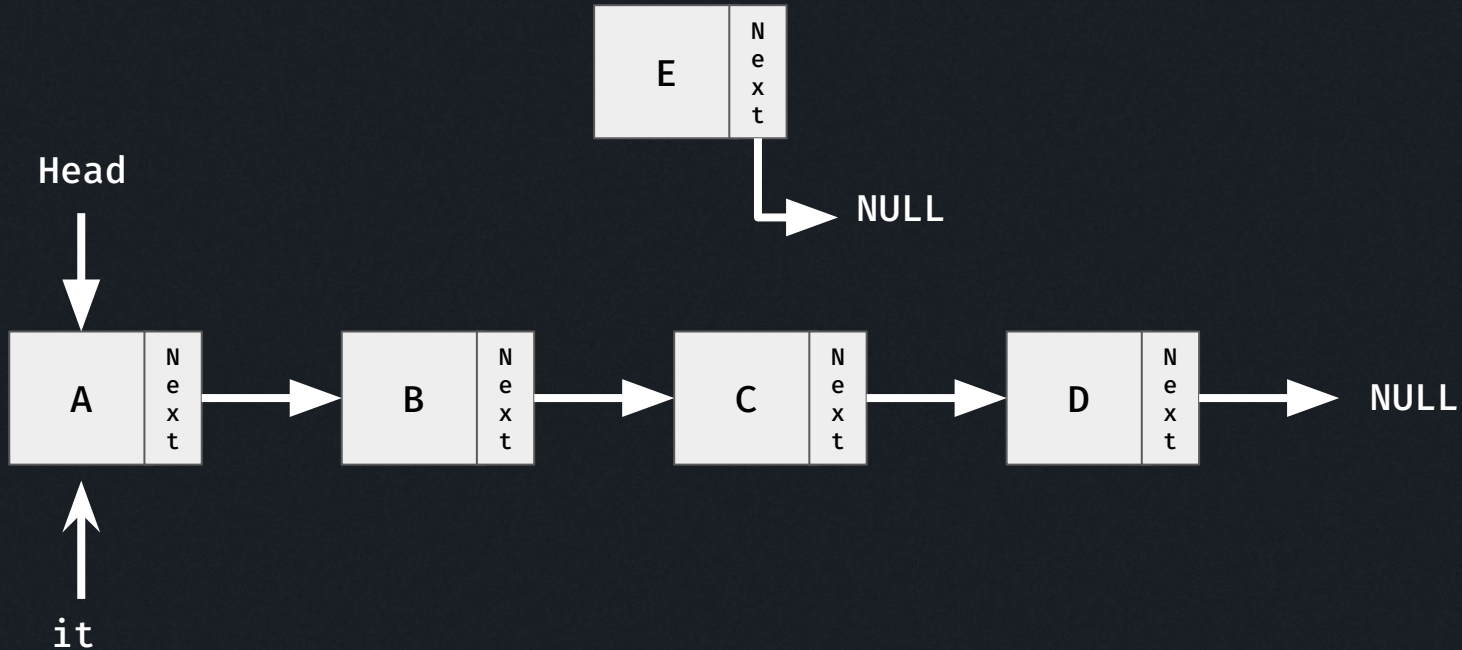
Inserção no fim

- Criar um novo nó
- Atualiza o último nó da lista apontar para o novo nó



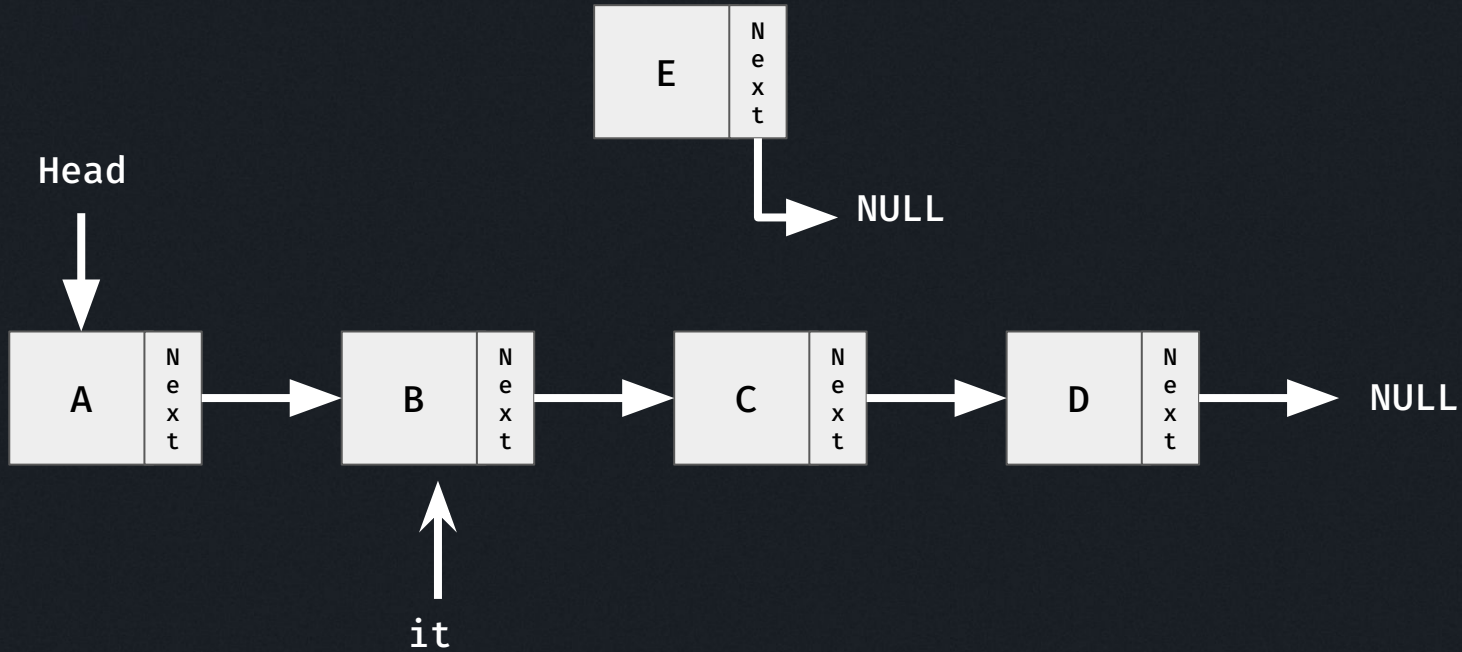
Inserção no fim

- Criar um novo nó
- Atualiza o último nó da lista apontar para o novo nó



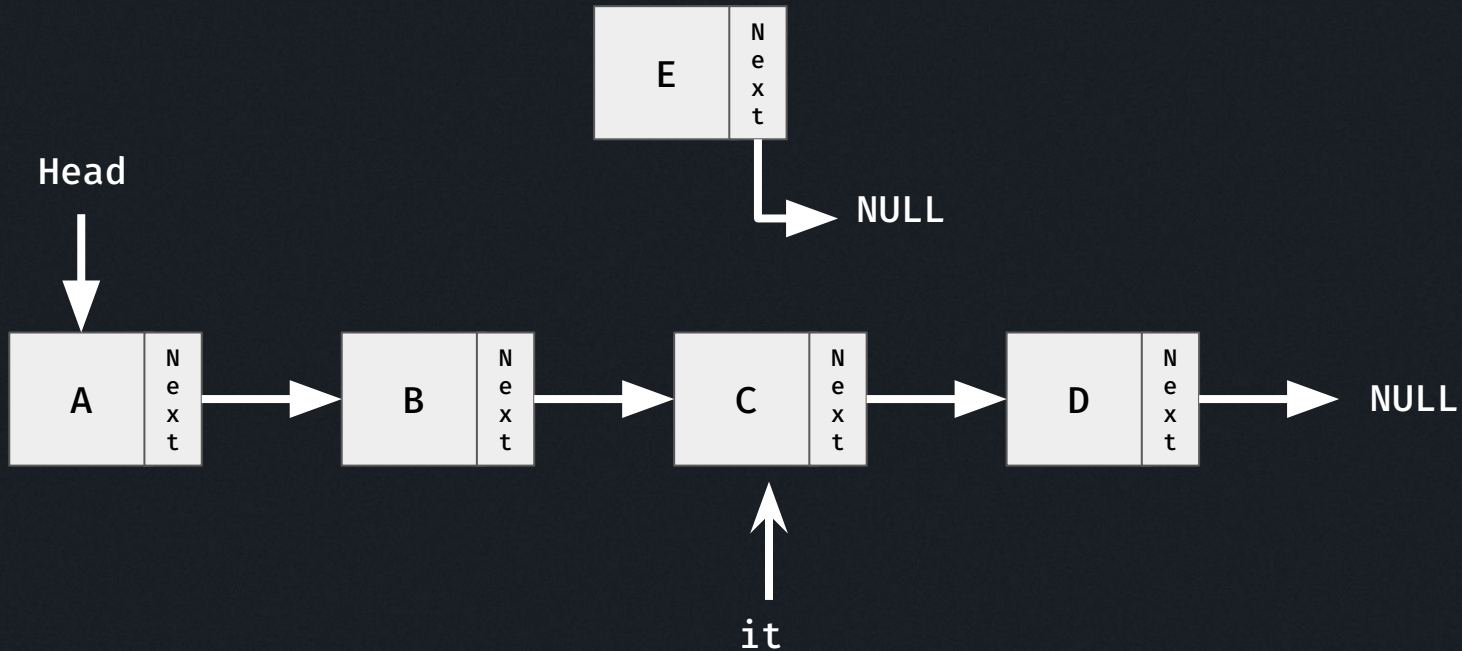
Inserção no fim

- Criar um novo nó
- Atualiza o último nó da lista apontar para o novo nó



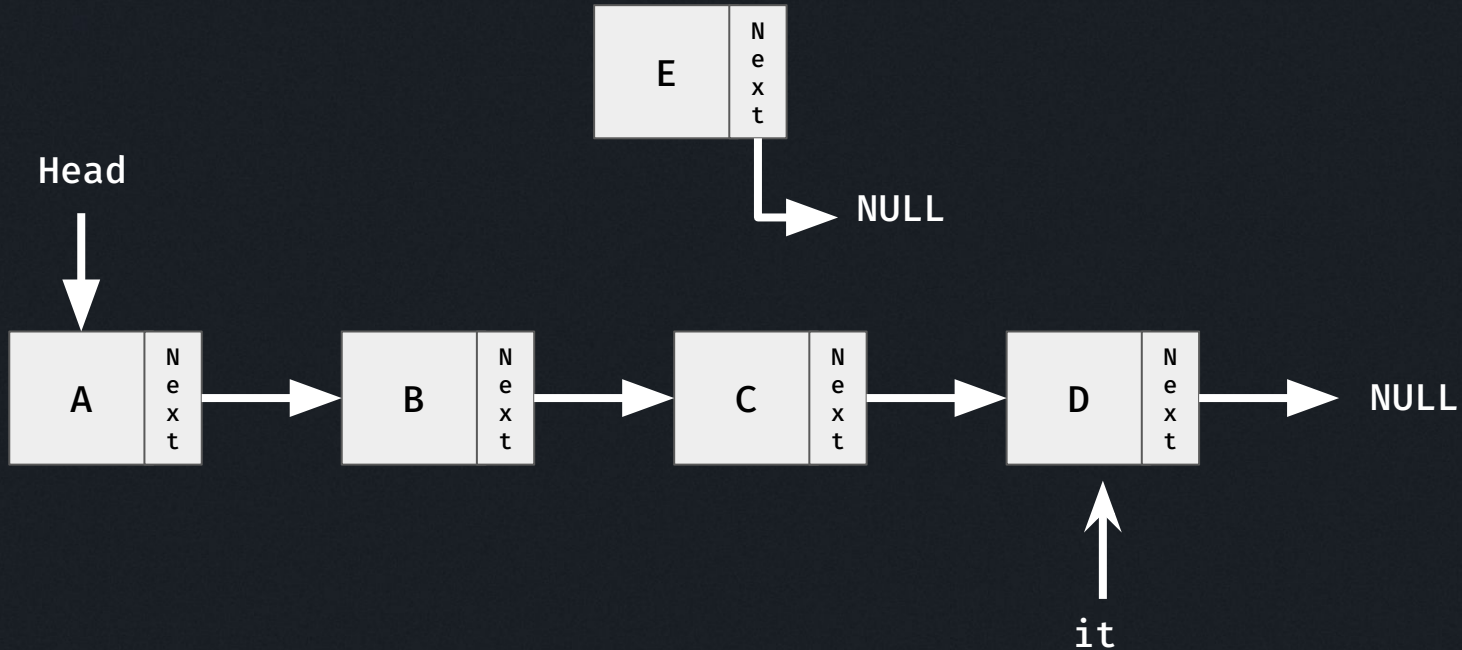
Inserção no fim

- Criar um novo nó
- Atualiza o último nó da lista apontar para o novo nó



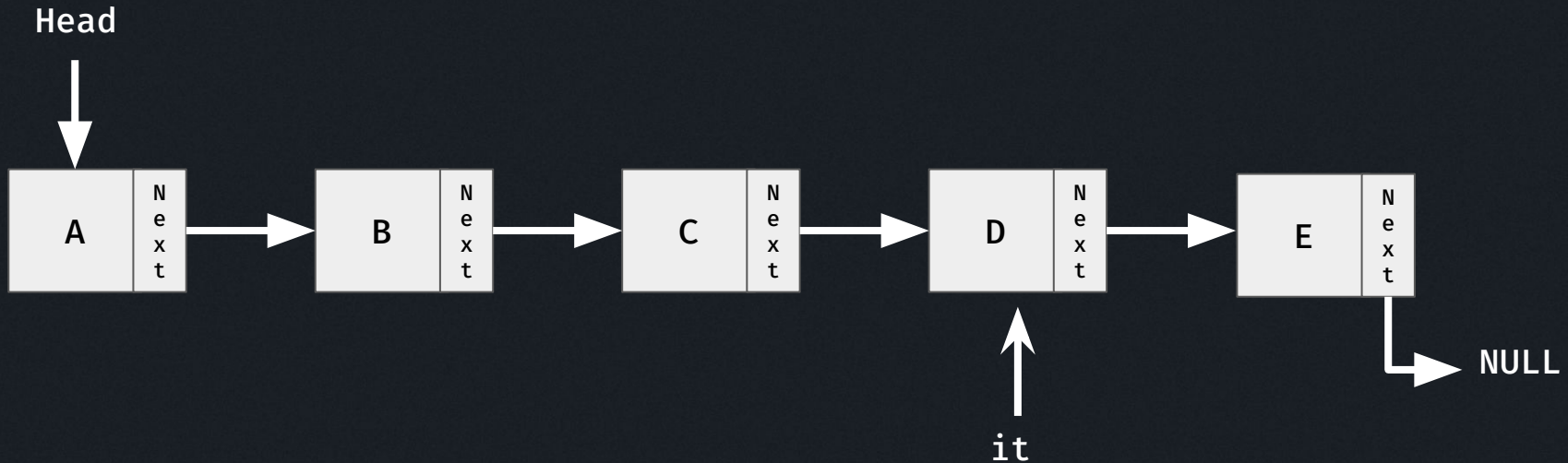
Inserção no fim

- Criar um novo nó
- Atualiza o último nó da lista apontar para o novo nó



Inserção no fim

- Criar um novo nó
- Atualiza o último nó da lista apontar para o novo nó



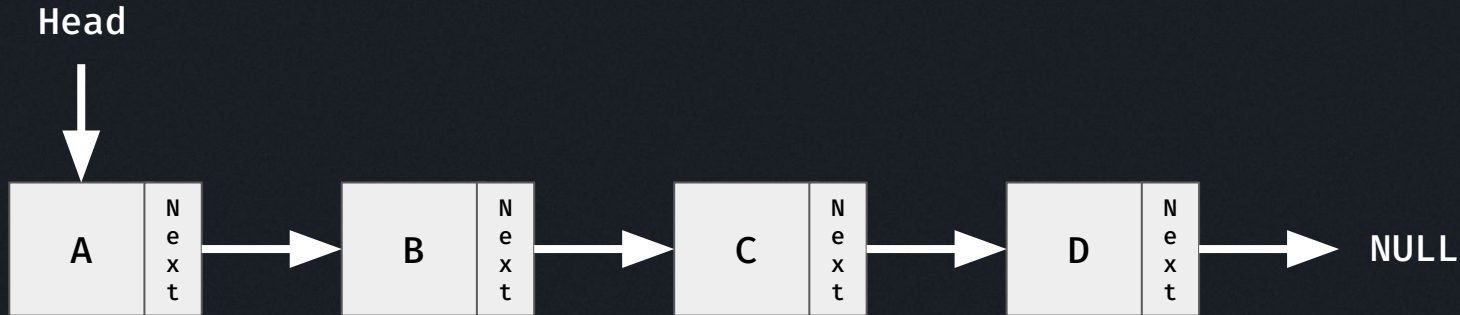
Inserção no fim

- Criar um novo nó
- Atualiza o último nó da lista apontar para o novo nó

```
insertEnd(data) {  
    const newNode = new Node(data);  
    if (this.head == null) {  
        this.head = newNode;  
        return;  
    }  
  
    let prev = this.head;  
    while (prev.next != null) {  
        prev = prev.next;  
    }  
    prev.next = newNode;  
}
```

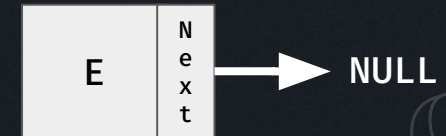
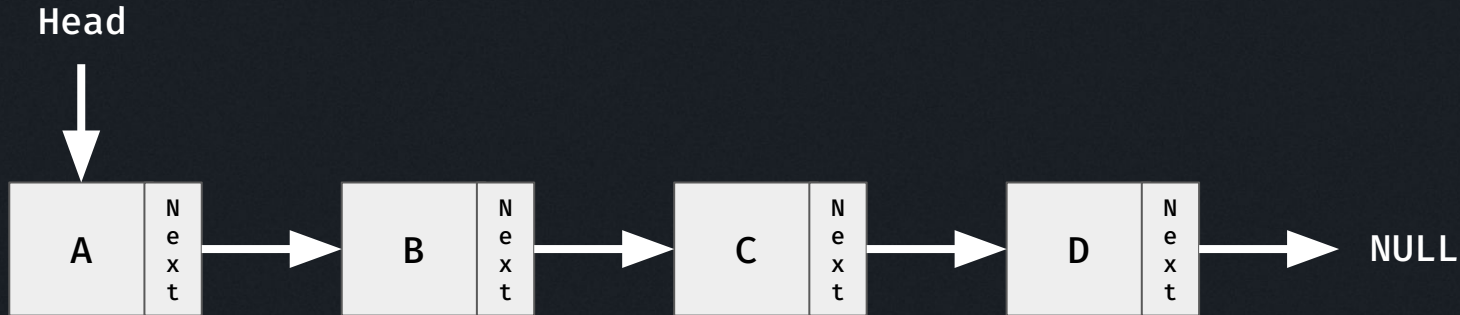
Inserção em uma posição específica

- Criar um novo nó
- Encontrar o nó na posição anterior a que queremos inserir (vamos chamá-lo de **q**)
- Atualizar a referência **next** do novo nó para apontar para o nó seguinte a **q**
- Atualizar a referência **next** do nó **q** para apontar para o novo nó



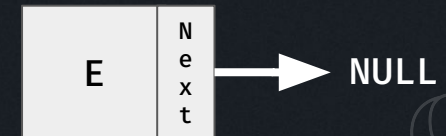
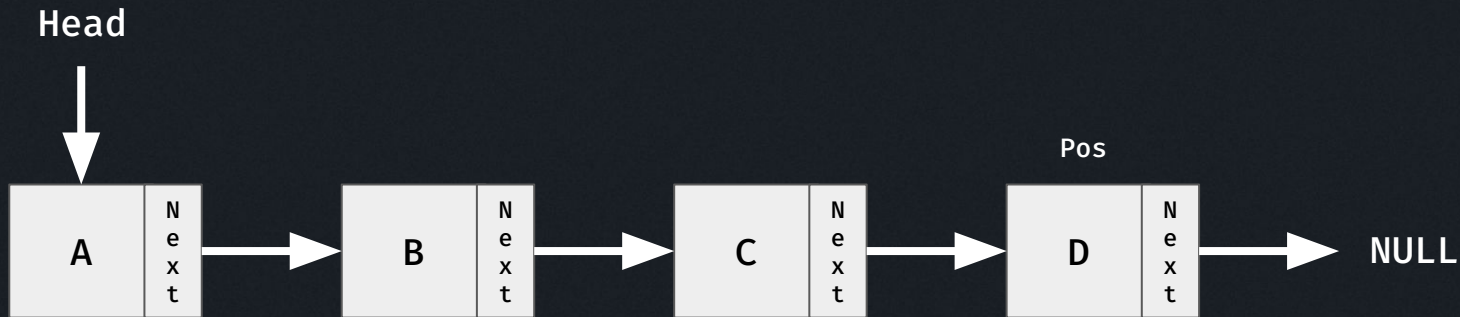
Inserção em uma posição específica

- Criar um novo nó
- Encontrar o nó na posição anterior a que queremos inserir (vamos chamá-lo de **q**)
- Atualizar a referência **next** do novo nó para apontar para o nó seguinte a **q**
- Atualizar a referência **next** do nó **q** para apontar para o novo nó



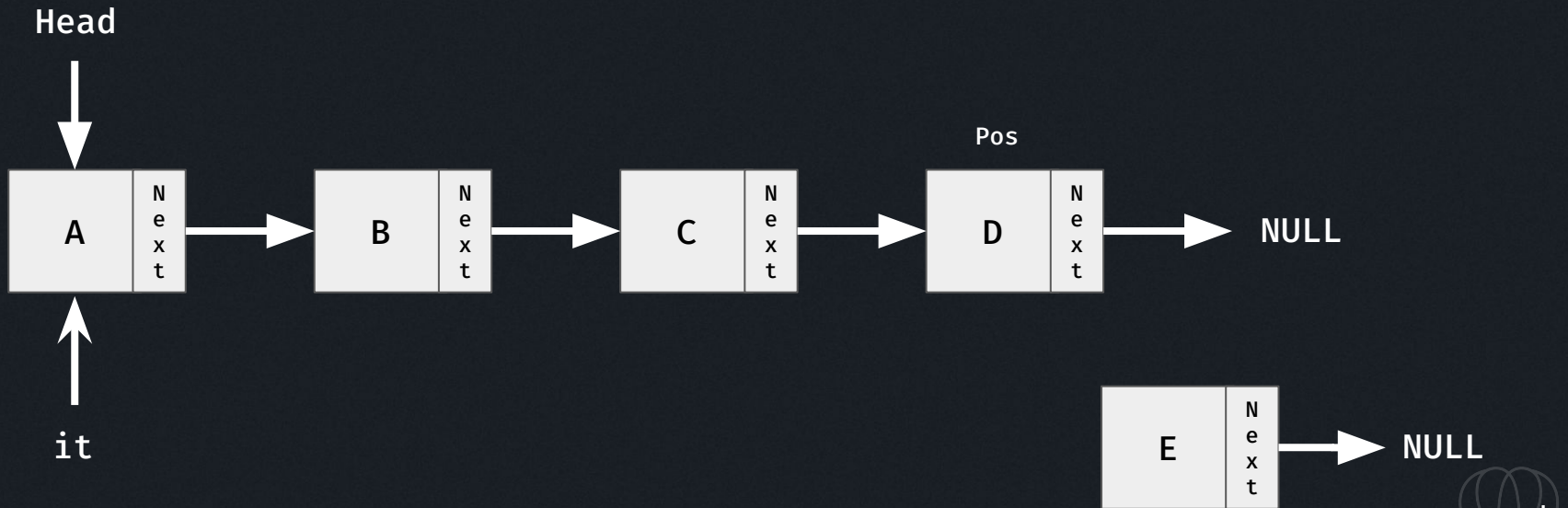
Inserção em uma posição específica

- Criar um novo nó
- Encontrar o nó na posição anterior a que queremos inserir (vamos chamá-lo de **q**)
- Atualizar a referência **next** do novo nó para apontar para o nó seguinte a **q**
- Atualizar a referência **next** do nó **q** para apontar para o novo nó



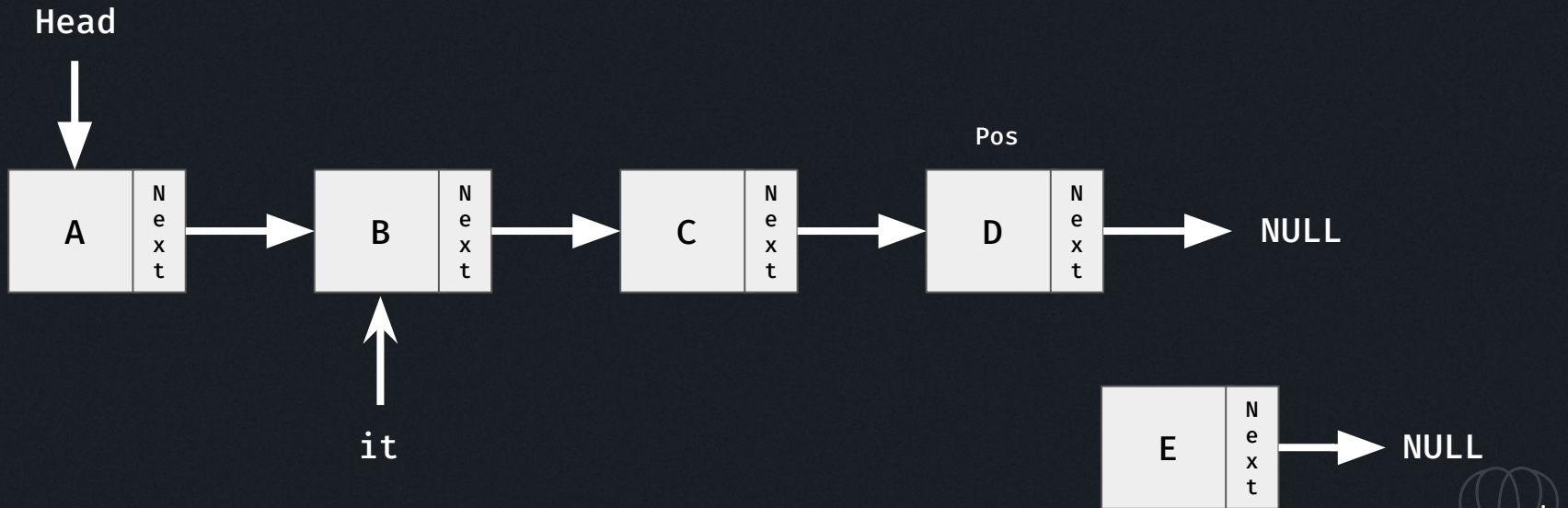
Inserção em uma posição específica

- Criar um novo nó
- Encontrar o nó na posição anterior a que queremos inserir (vamos chamá-lo de **q**)
- Atualizar a referência **next** do novo nó para apontar para o nó seguinte a **q**
- Atualizar a referência **next** do nó **q** para apontar para o novo nó



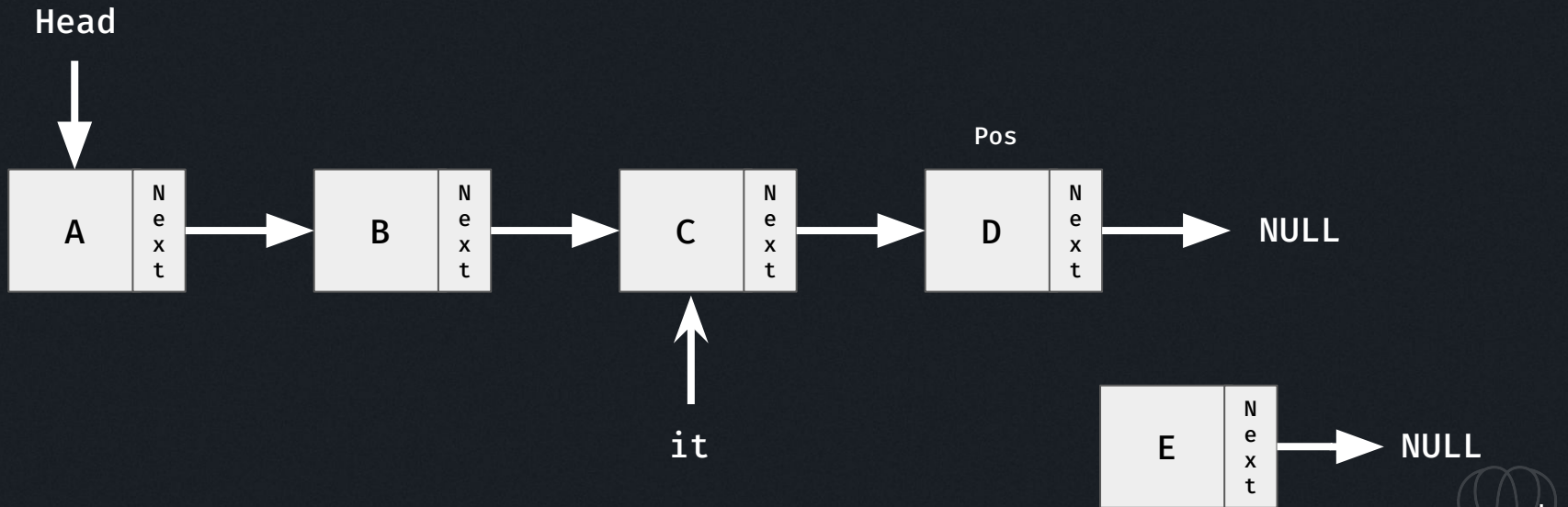
Inserção em uma posição específica

- Criar um novo nó
- Encontrar o nó na posição anterior a que queremos inserir (vamos chamá-lo de **q**)
- Atualizar a referência **next** do novo nó para apontar para o nó seguinte a **q**
- Atualizar a referência **next** do nó **q** para apontar para o novo nó



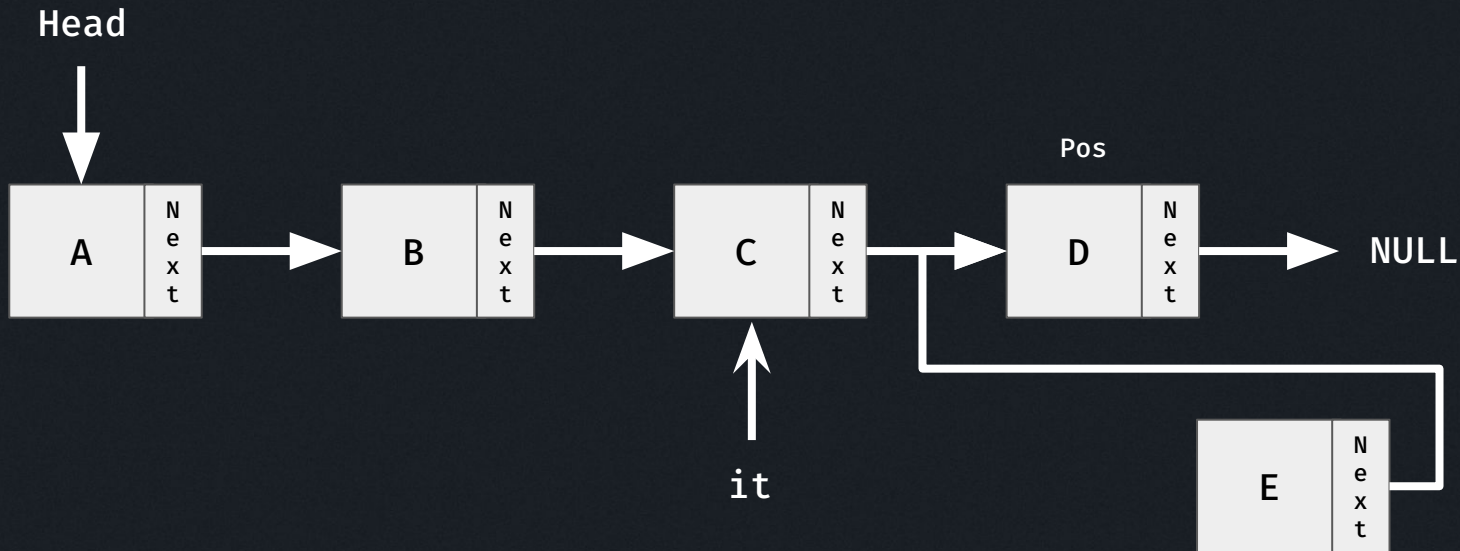
Inserção em uma posição específica

- Criar um novo nó
- Encontrar o nó na posição anterior a que queremos inserir (vamos chamá-lo de **q**)
- Atualizar a referência **next** do novo nó para apontar para o nó seguinte a **q**
- Atualizar a referência **next** do nó **q** para apontar para o novo nó



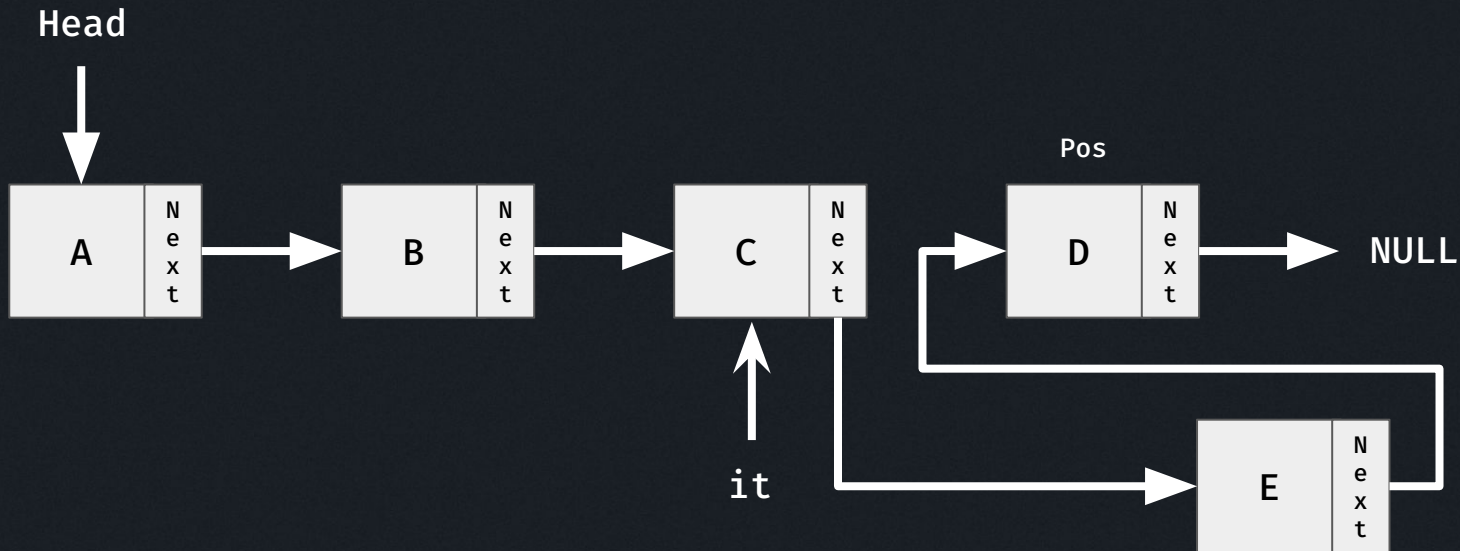
Inserção em uma posição específica

- Criar um novo nó
- Encontrar o nó na posição anterior a que queremos inserir (vamos chamá-lo de **q**)
- Atualizar a referência **next** do novo nó para apontar para o nó seguinte a **q**
- Atualizar a referência **next** do nó **q** para apontar para o novo nó



Inserção em uma posição específica

- Criar um novo nó
- Encontrar o nó na posição anterior a que queremos inserir (vamos chamá-lo de **q**)
- Atualizar a referência **next** do novo nó para apontar para o nó seguinte a **q**
- Atualizar a referência **next** do nó **q** para apontar para o novo nó



Inserção em uma posição específica

- Criar um novo nó
- Encontrar o nó na posição anterior a que queremos inserir (vamos chamá-lo de **q**)
- Atualizar a referência **next** do novo nó para apontar para o nó seguinte a **q**
- Atualizar a referência **next** do nó **q** para apontar para o novo nó

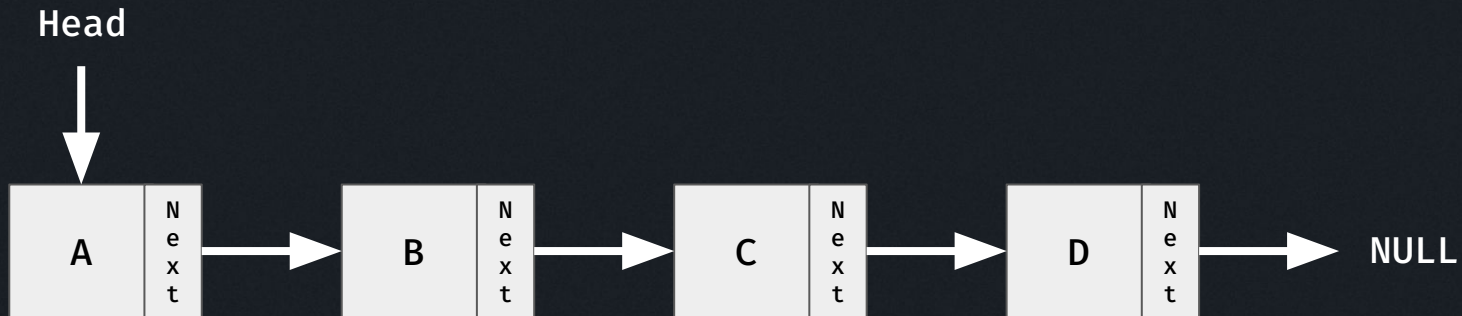
```
insertAfter(data, pos) {  
    const newNode = new Node(data);  
    let prev = this.head;  
    for (let i = 0; i < pos; i++){  
        if (prev == null) return;  
        prev = prev.next;  
    }  
    newNode.next = prev.next;  
    prev.next = newNode;  
}
```

Remoção

- **3 Casos:**
 - Remoção no início
 - Remoção no fim
 - Remoção em uma posição específica

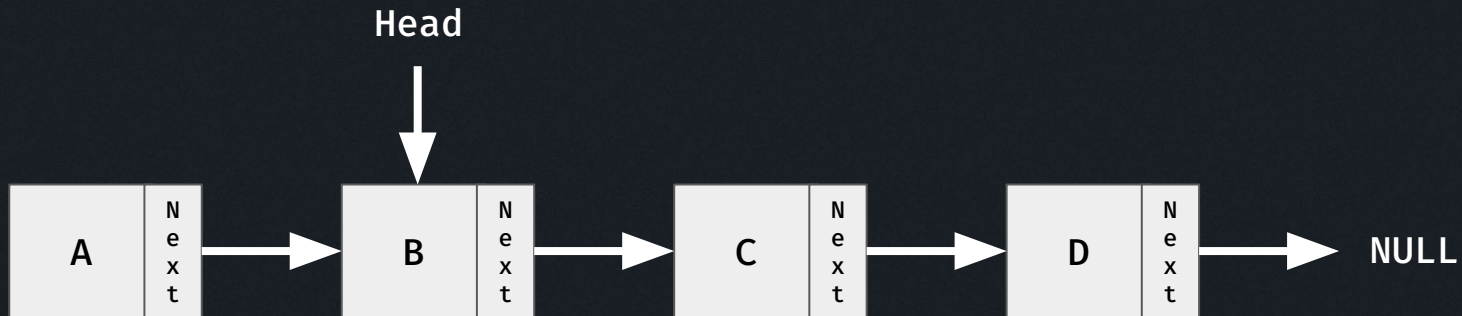
Remoção no início

- Atualizar a referência **head** para apontar para o segundo nó da lista (**head.next**)



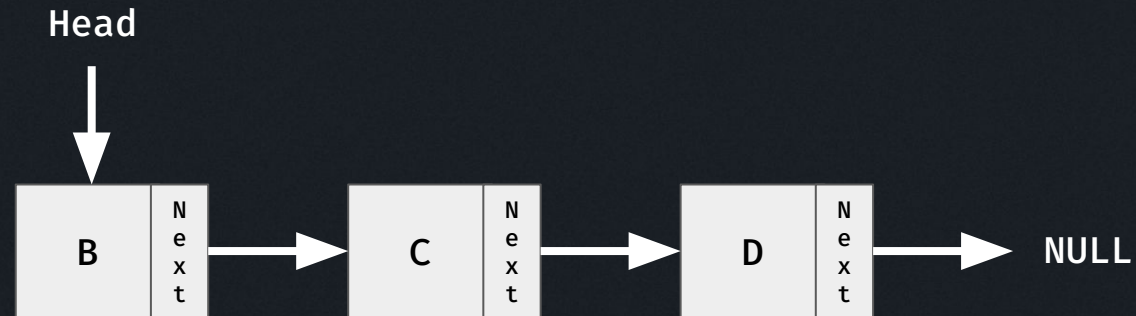
Remoção no início

- Atualizar a referência **head** para apontar para o segundo nó da lista (**head.next**)



Remoção no início

- Atualizar a referência **head** para apontar para o segundo nó da lista (**head.next**)



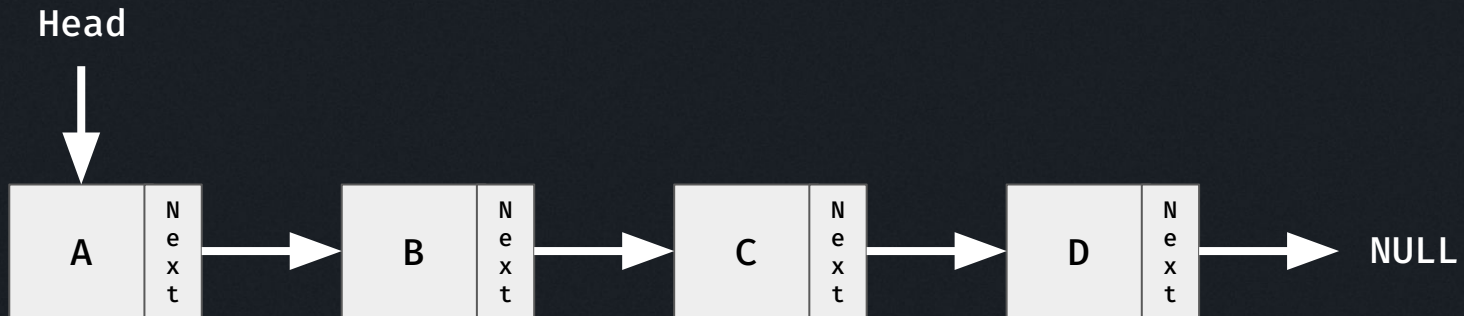
Remoção no início

- Atualizar a referência **head** para apontar para o segundo nó da lista (**head.next**)

```
removeBegin() {  
    if (this.head == null) return;  
    this.head = this.head.next;  
}
```

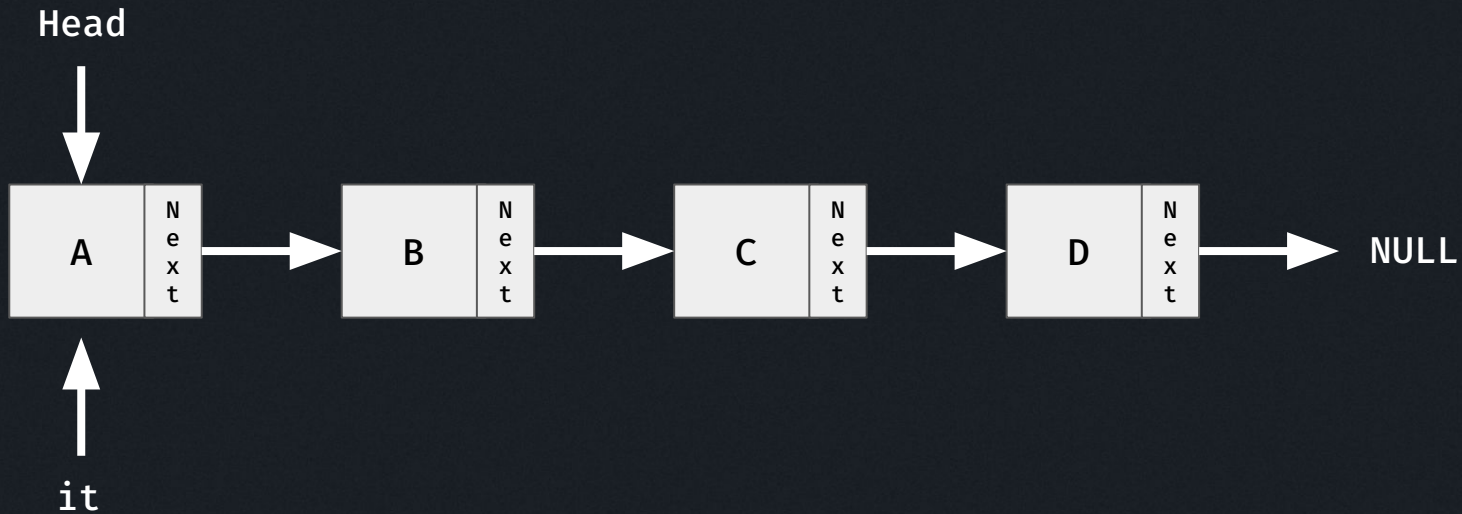
Remoção no fim

- Atualiza a referência **next** do penúltimo nó para apontar para **null**



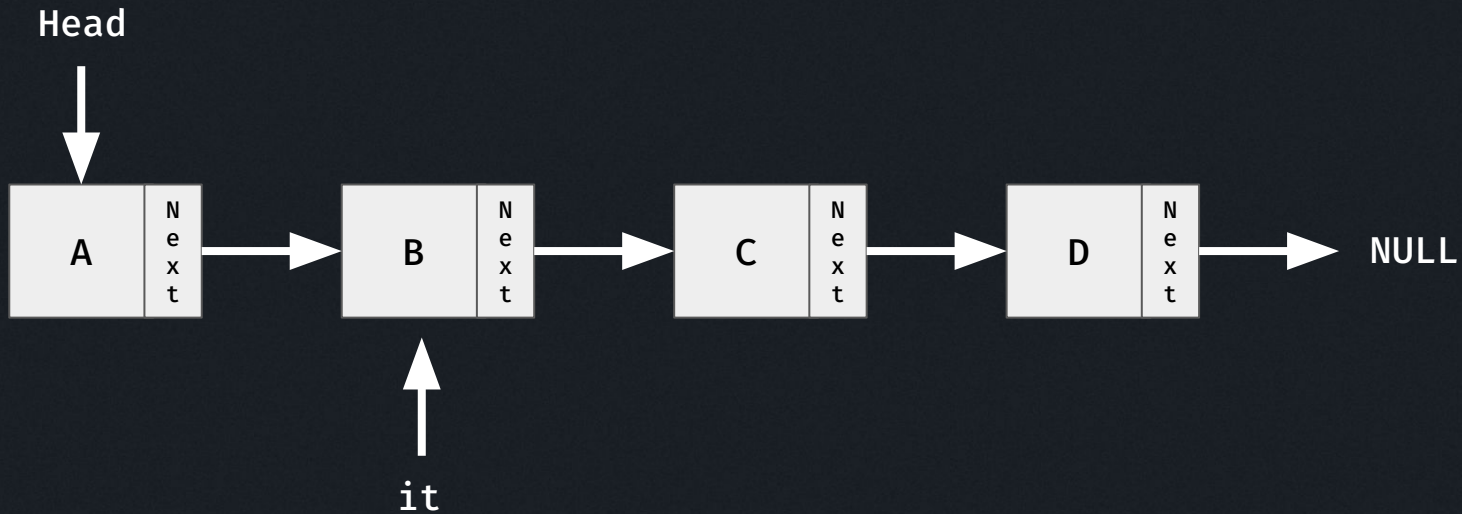
Remoção no fim

- Atualiza a referência **next** do penúltimo nó para apontar para **null**



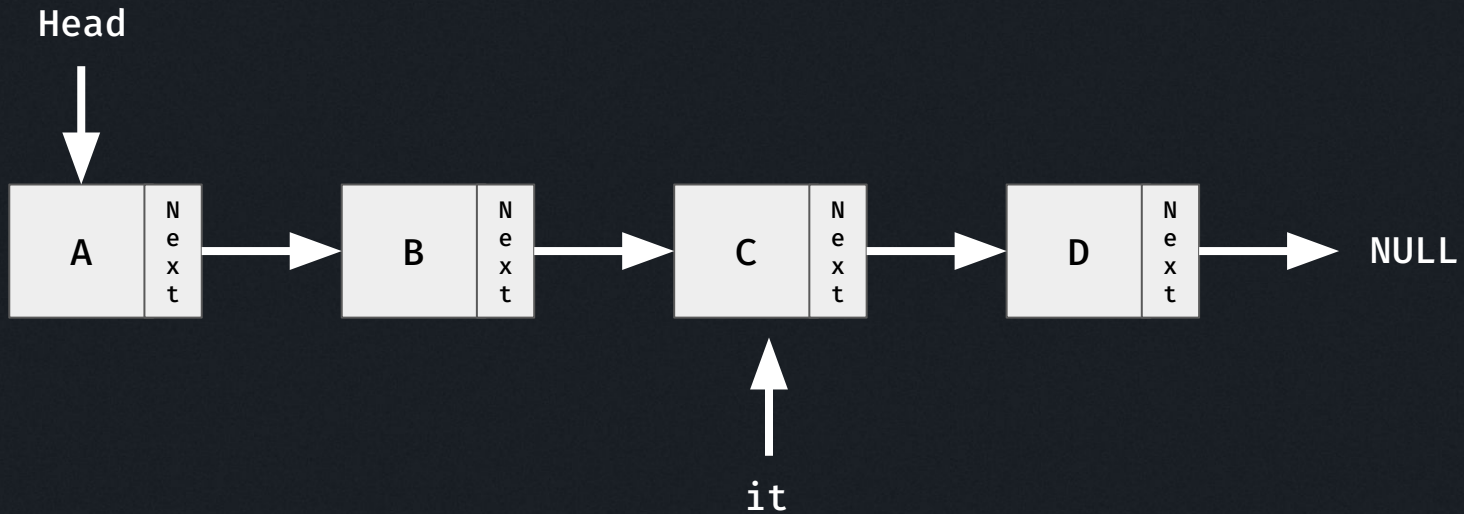
Remoção no fim

- Atualiza a referência **next** do penúltimo nó para apontar para **null**



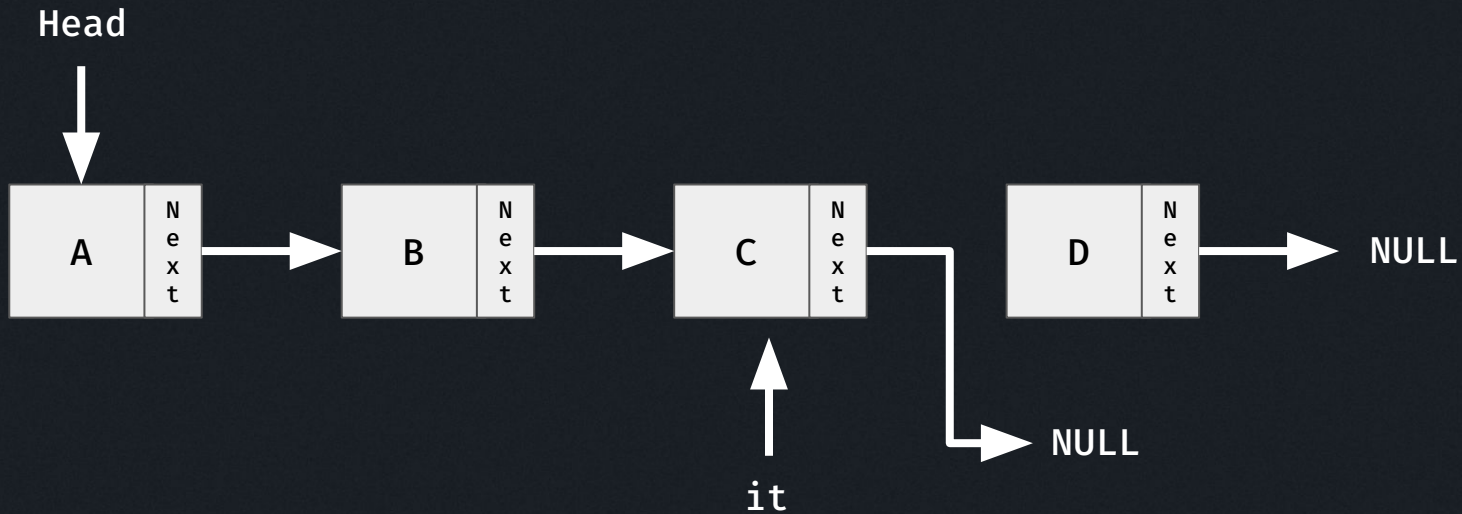
Remoção no fim

- Atualiza a referência **next** do penúltimo nó para apontar para **null**



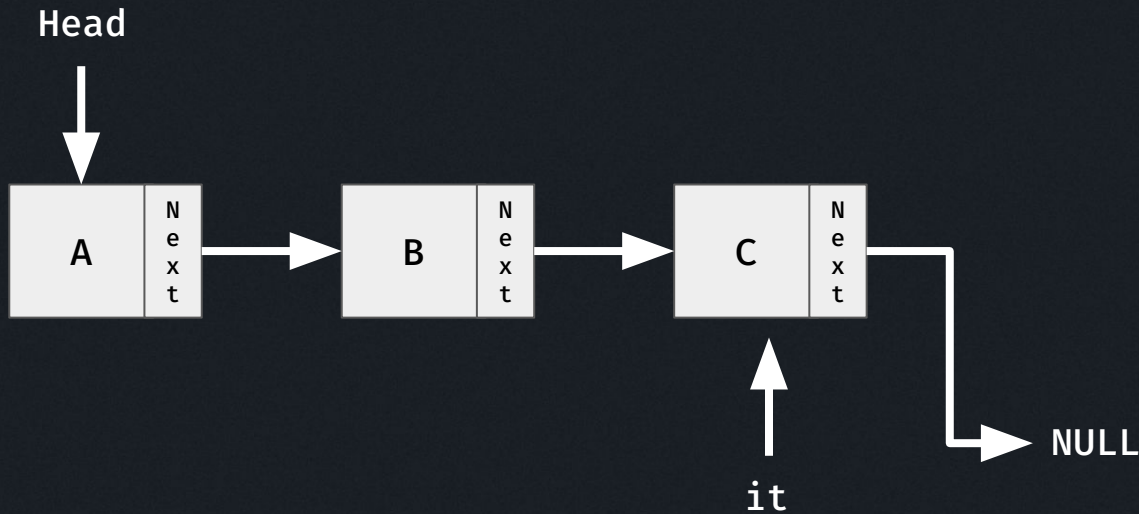
Remoção no fim

- Atualiza a referência **next** do penúltimo nó para apontar para **null**



Remoção no fim

- Atualiza a referência **next** do penúltimo nó para apontar para **null**



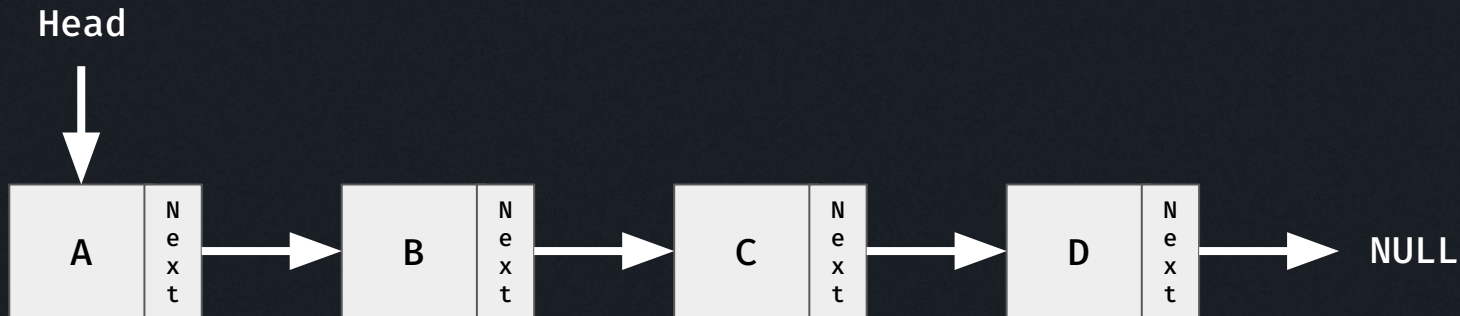
Remoção no fim

- Atualiza a referência **next** do penúltimo nó para apontar para **null**

```
removeEnd() {  
    if (this.head == null) return;  
    if (this.head.next == null) {  
        this.head = null;  
        return;  
    }  
    let prev = this.head;  
    while (prev.next.next != null) {  
        prev = prev.next;  
    }  
    prev.next = null;  
}
```

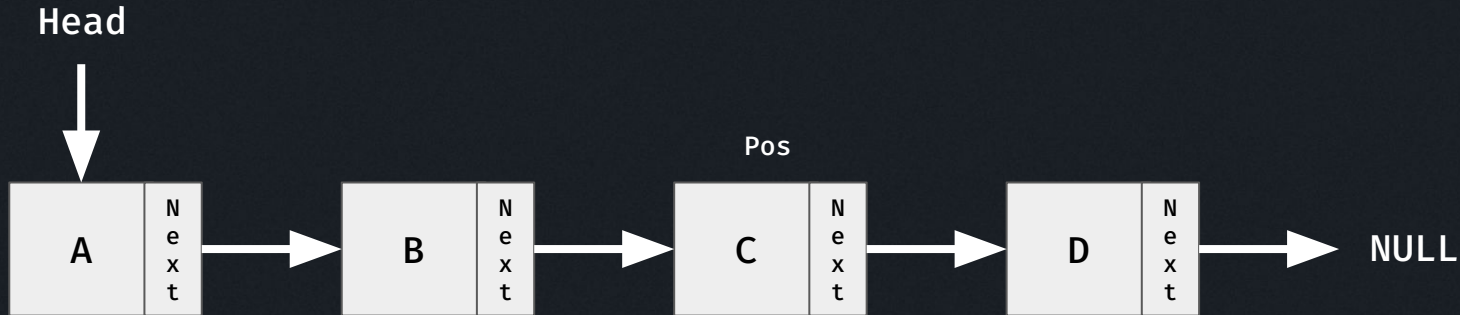
Remoção em uma posição específica

- Encontrar o nó na posição que queremos deletar (vamos chamá-lo de **q**)
- Atualiza a referência **next** do nó **anterior** à **q** para apontar para **q.next**



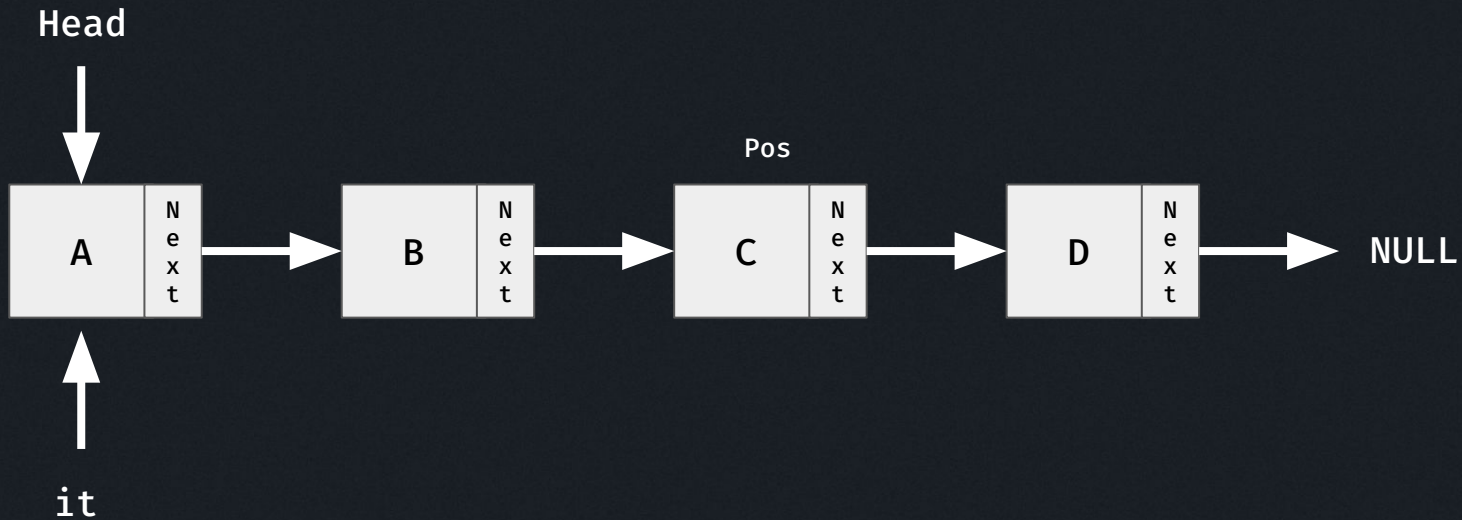
Remoção em uma posição específica

- Encontrar o nó na posição anterior a que queremos deletar (vamos chamá-lo de **q**)
- Atualiza a referência **next** do nó **anterior** à **q** para apontar para **q.next**



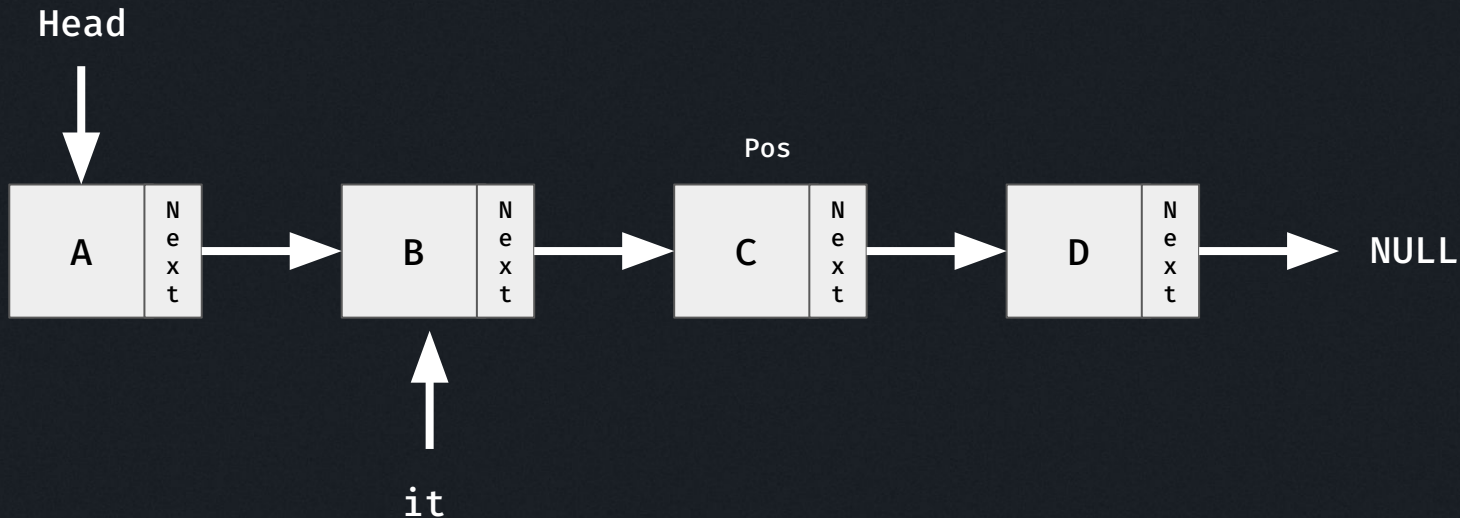
Remoção em uma posição específica

- Encontrar o nó na posição anterior a que queremos deletar (vamos chamá-lo de **q**)
- Atualiza a referência **next** do nó **anterior** à **q** para apontar para **q.next**



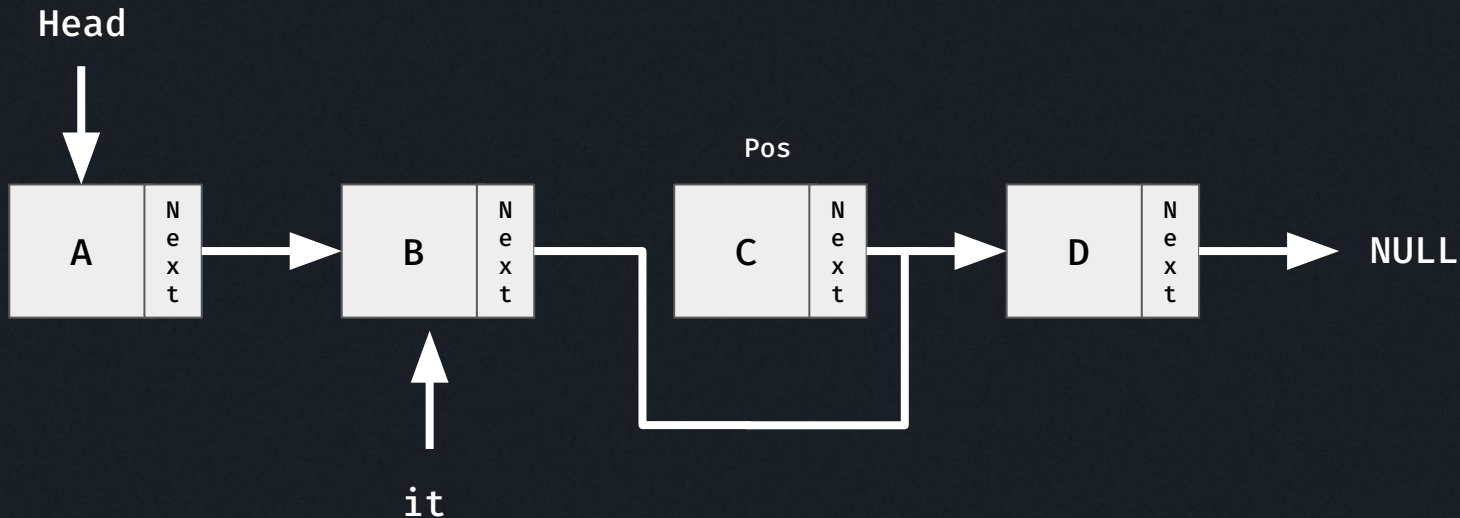
Remoção em uma posição específica

- Encontrar o nó na posição anterior a que queremos deletar (vamos chamá-lo de **q**)
- Atualiza a referência **next** do nó **anterior** à **q** para apontar para **q.next**



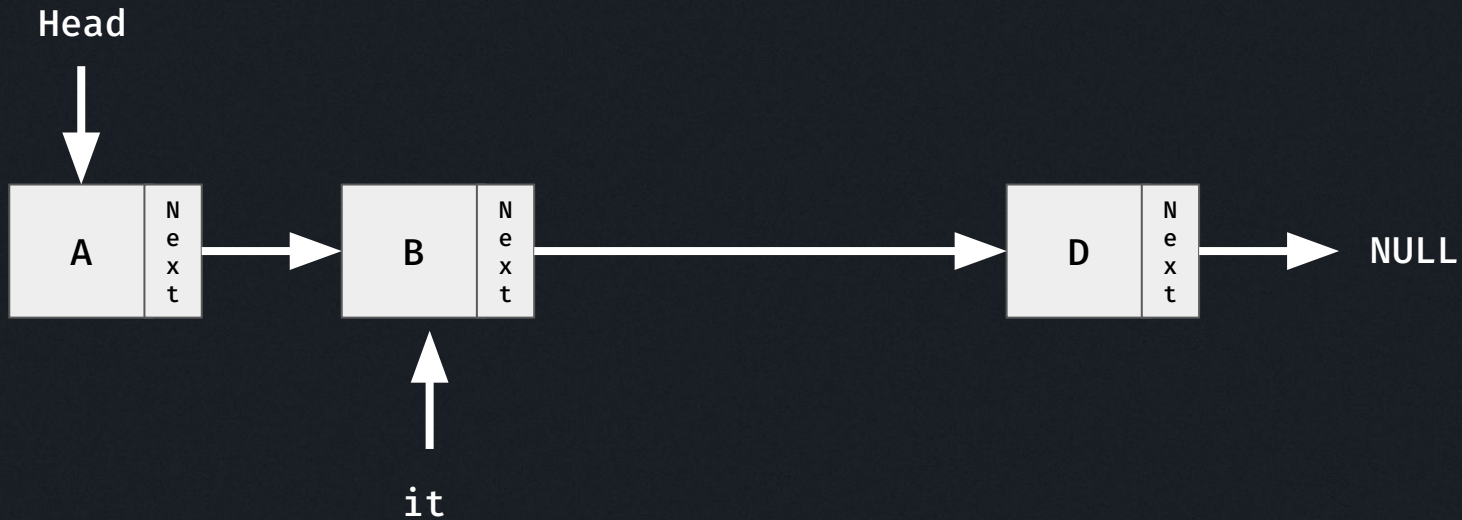
Remoção em uma posição específica

- Encontrar o nó na posição anterior a que queremos deletar (vamos chamá-lo de **q**)
- Atualiza a referência **next** do nó **anterior** à **q** para apontar para **q.next**



Remoção em uma posição específica

- Encontrar o nó na posição anterior a que queremos deletar (vamos chamá-lo de **q**)
- Atualiza a referência **next** do nó **anterior** à **q** para apontar para **q.next**



Remoção em uma posição específica

```
removePos(pos) {  
    if (this.head == null) return;  
    let prev = this.head;  
    for (let i = 1; i < pos; i++) {  
        if (prev == null) return;  
        prev = prev.next;  
    }  
    if (prev.next == null) return;  
  
    prev.next = prev.next.next;  
}
```

Operações

- Criação da lista
- Inserção
- Remoção
- **Busca**
- **Inversão**

Middle of the Linked List

<https://leetcode.com/problems/middle-of-the-linked-list/>

876. Middle of the Linked List

Solved 

Easy  Topics  Companies

Given the `head` of a singly linked list, return *the middle node of the linked list*.

If there are two middle nodes, return **the second middle node**.

Example 1:

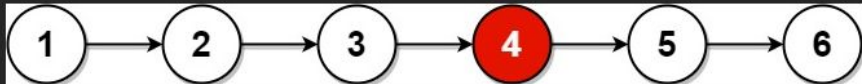


Input: `head = [1,2,3,4,5]`

Output: `[3,4,5]`

Explanation: The middle node of the list is node 3.

Example 2:



Input: `head = [1,2,3,4,5,6]`

Output: `[4,5,6]`

Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

Constraints:

- The number of nodes in the list is in the range `[1, 100]`.
- `1 <= Node.val <= 100`

Middle of the Linked List

<https://leetcode.com/problems/middle-of-the-linked-list/>

876. Middle of the Linked List

Solved 

Easy  Topics  Companies

Given the `head` of a singly linked list, return *the middle node of the linked list*.

If there are two middle nodes, return **the second middle node**.

Example 1:



Input: `head = [1,2,3,4,5]`

Output: `[3,4,5]`

Explanation: The middle node of the list is node 3.

Example 2:



Input: `head = [1,2,3,4,5,6]`

Output: `[4,5,6]`

Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

Constraints:

- The number of nodes in the list is in the range `[1, 100]`.
- `1 <= Node.val <= 100`

Estratégia: contar quantos elementos existem na lista.

Iterar a lista até a metade e retornar o nó do meio

Middle of the Linked List

<https://leetcode.com/problems/middle-of-the-linked-list/>



```
ListNode MiddleNode(ListNode head) {  
    var count = 0;  
    var it = head;  
    while (it != null) {  
        count ++;  
        it = it.next;  
    }  
  
    var middle = count / 2;  
    ListNode output = head;  
    for (int i = 1; i ≤ middle; i++) {  
        output = output.next;  
    }  
  
    return output;  
}
```

Middle of the Linked List

<https://leetcode.com/problems/middle-of-the-linked-list/>

876. Middle of the Linked List

Solved 

Easy  Topics  Companies

Given the `head` of a singly linked list, return *the middle node of the linked list*.

If there are two middle nodes, return **the second middle node**.

Example 1:

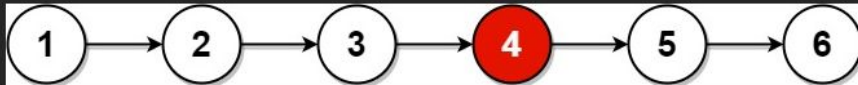


Input: `head = [1,2,3,4,5]`

Output: `[3,4,5]`

Explanation: The middle node of the list is node 3.

Example 2:



Input: `head = [1,2,3,4,5,6]`

Output: `[4,5,6]`

Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

Constraints:

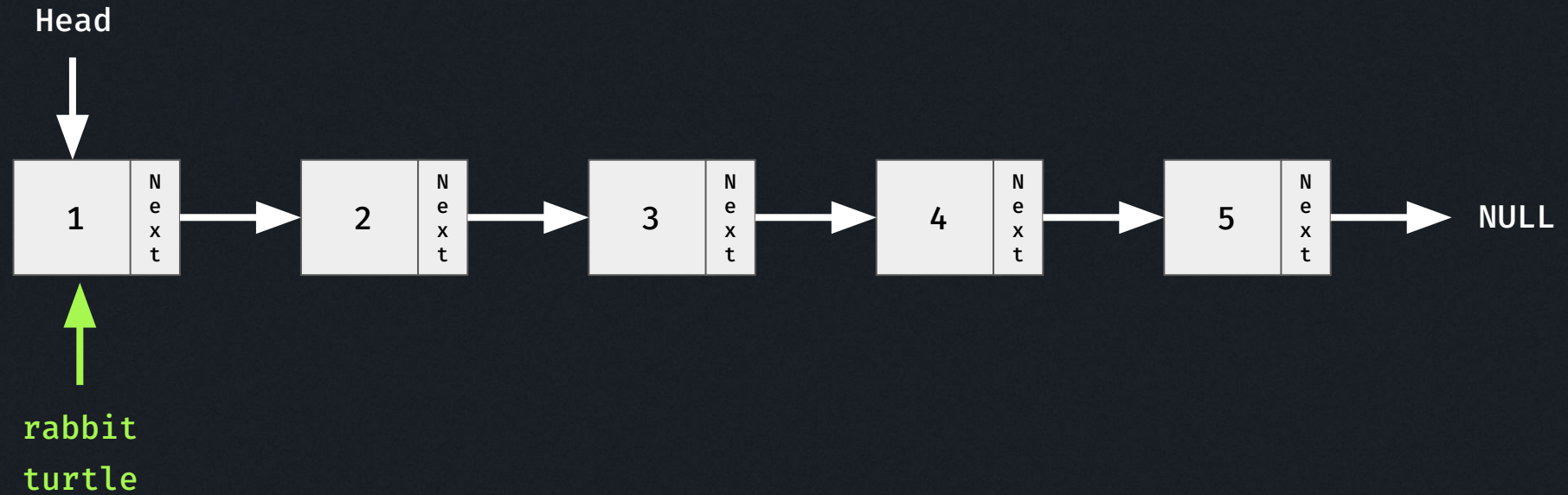
- The number of nodes in the list is in the range `[1, 100]`.
- `1 <= Node.val <= 100`

Estratégia 2: a lebre e a tartaruga.

Criar um ponteiro para lista que anda de nó em nó (**tartaruga**) e outro que anda dois nós por vez (**lebre**). Quando o ponteiro lebre chegar no final da lista, o ponteiro tartaruga estará no meio.

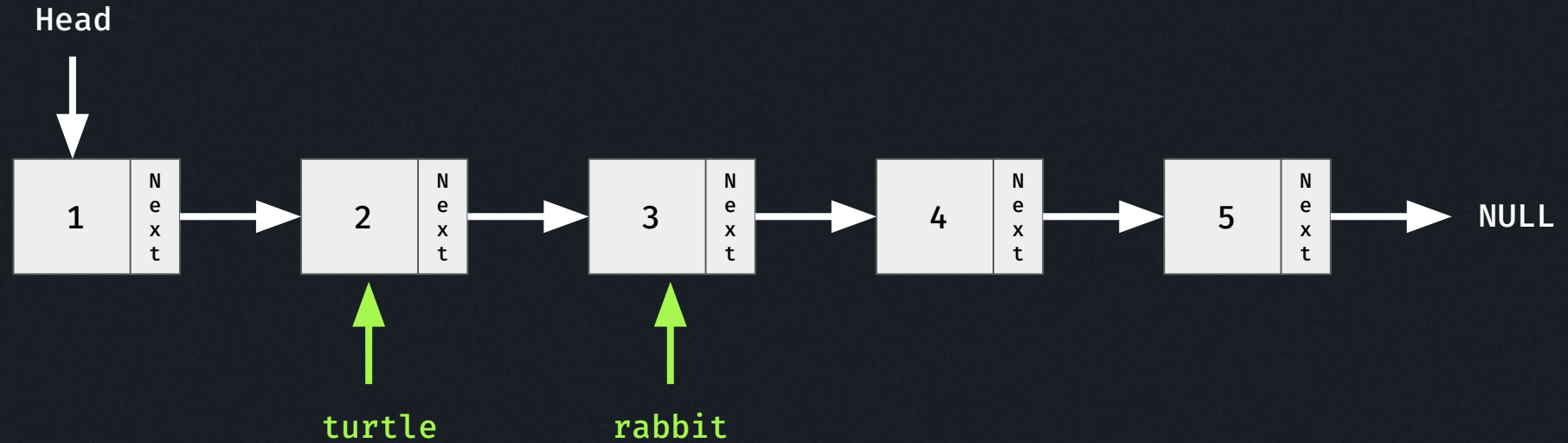
Middle of the Linked List

<https://leetcode.com/problems/middle-of-the-linked-list/>



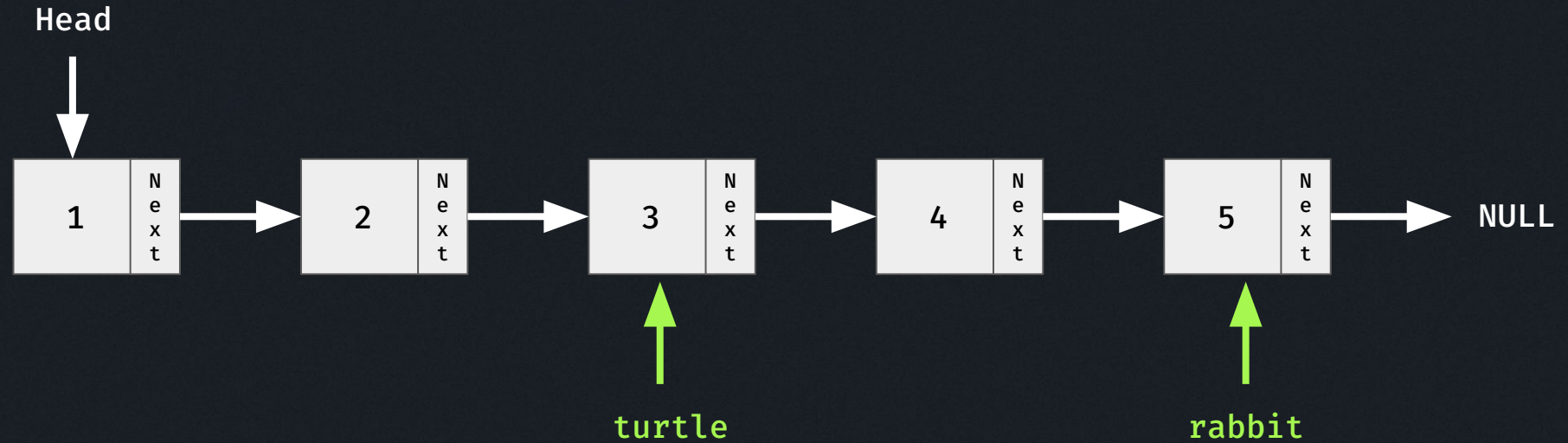
Middle of the Linked List

<https://leetcode.com/problems/middle-of-the-linked-list/>



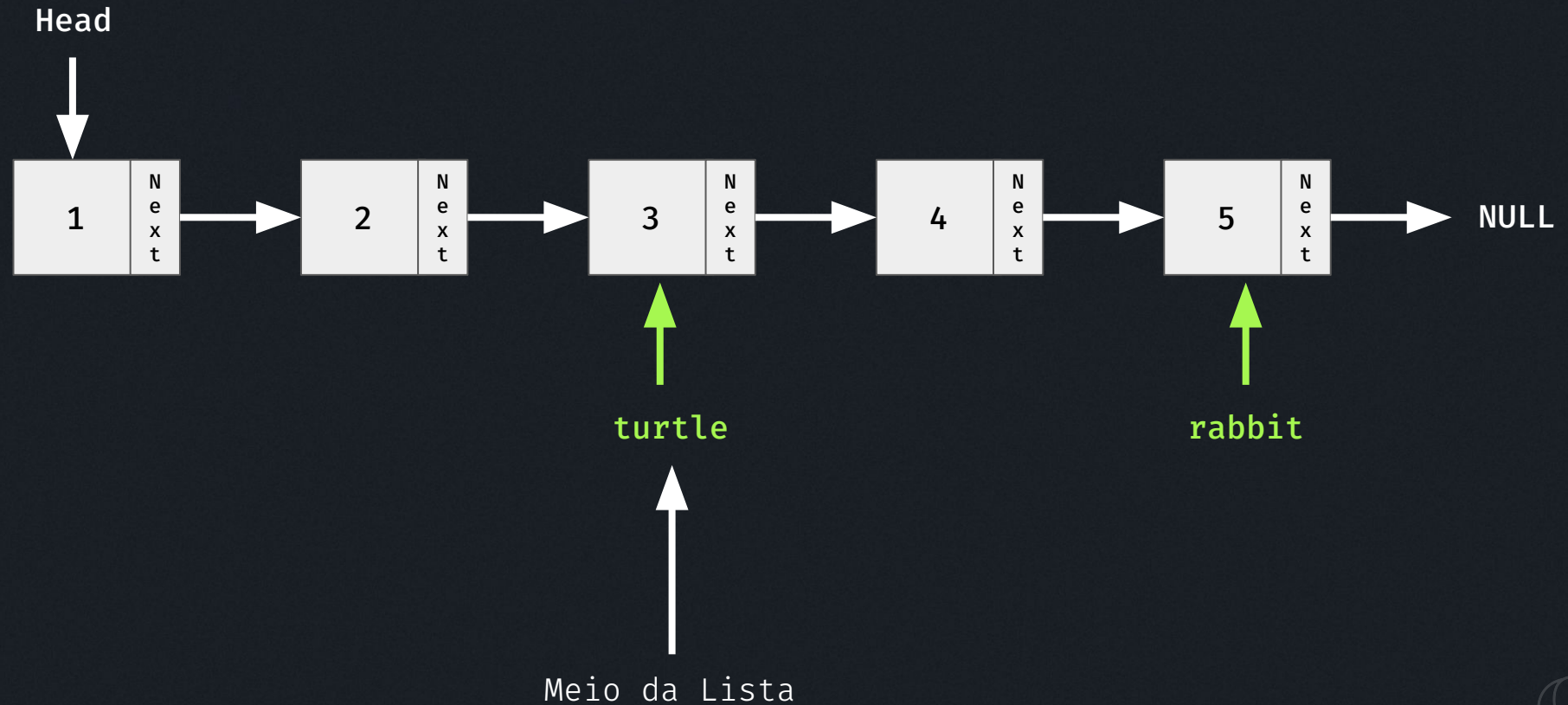
Middle of the Linked List

<https://leetcode.com/problems/middle-of-the-linked-list/>



Middle of the Linked List

<https://leetcode.com/problems/middle-of-the-linked-list/>



Middle of the Linked List

<https://leetcode.com/problems/middle-of-the-linked-list/>

```
ListNode MiddleNode(ListNode head) {  
    var rabbit = head;  
    var turtle = head;  
    while (rabbit != null && rabbit.next != null) {  
        turtle = turtle.next;  
        rabbit = rabbit.next.next;  
    }  
    return turtle;  
}
```

Reverse Linked List

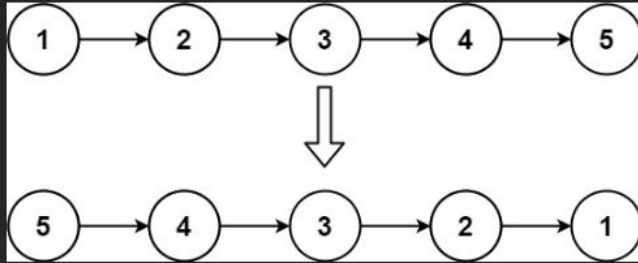
<https://leetcode.com/problems/reverse-linked-list/>

206. Reverse Linked List

Easy Topics Companies

Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

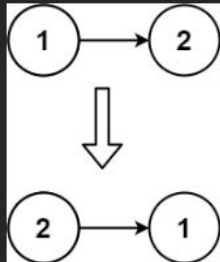
Example 1:



Input: head = [1,2,3,4,5]

Output: [5,4,3,2,1]

Example 2:



Input: head = [1,2]

Output: [2,1]

Reverse Linked List

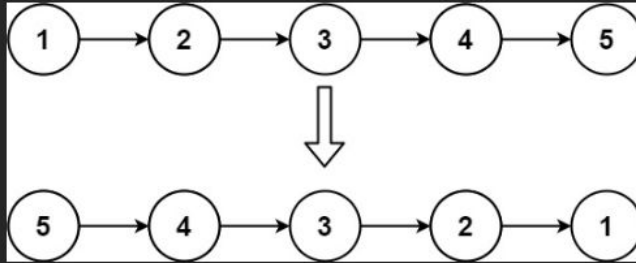
<https://leetcode.com/problems/reverse-linked-list/>

206. Reverse Linked List

Easy Topics Companies

Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

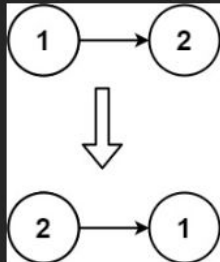
Example 1:



Input: head = [1,2,3,4,5]

Output: [5,4,3,2,1]

Example 2:



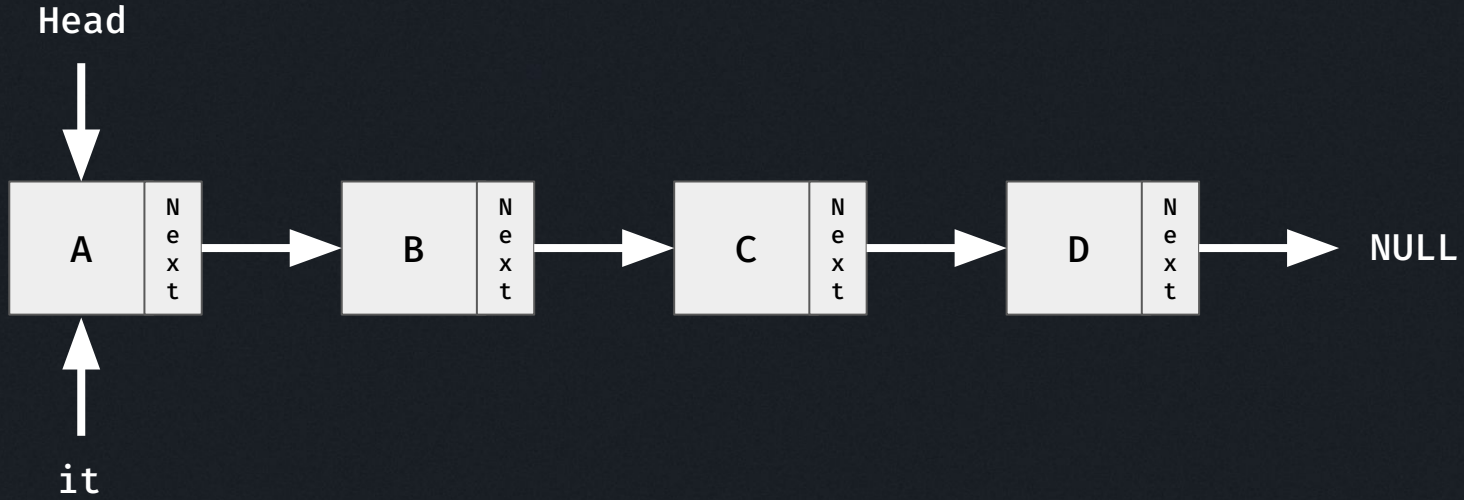
Input: head = [1,2]

Output: [2,1]

Estratégia: iterar sobre a lista e a cada passo fazer o nó atual apontar para o nó anterior

Reverse Linked List

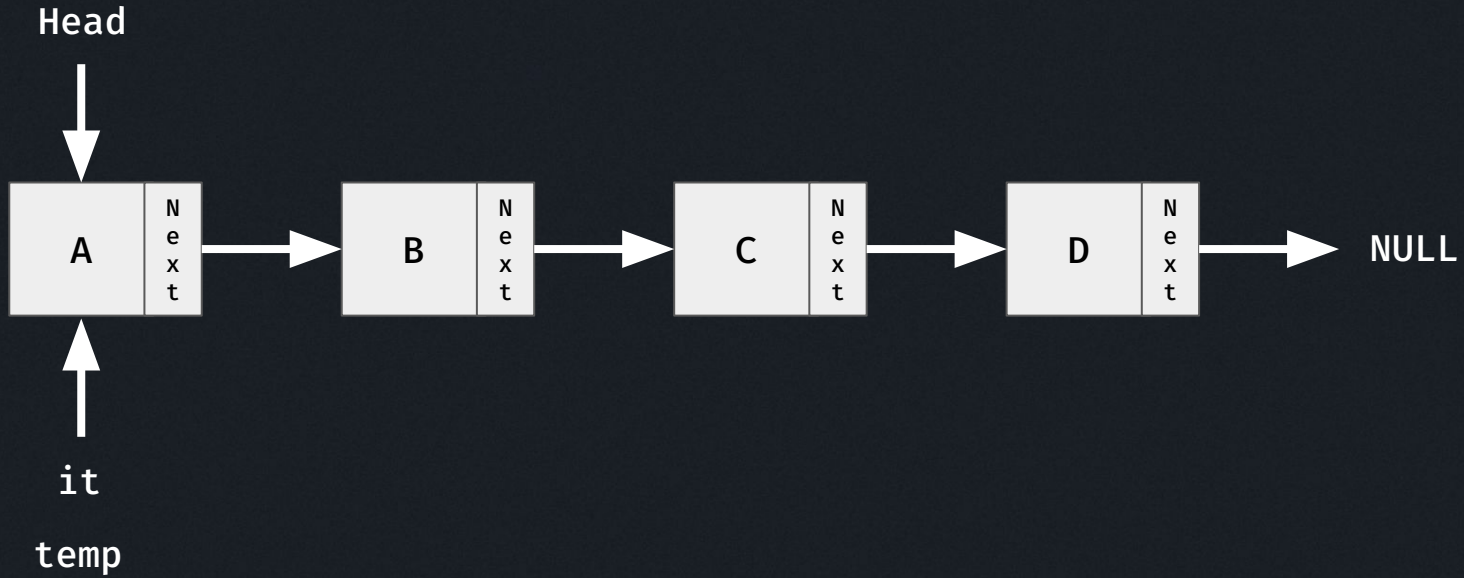
<https://leetcode.com/problems/reverse-linked-list/>



Prev → NULL

Reverse Linked List

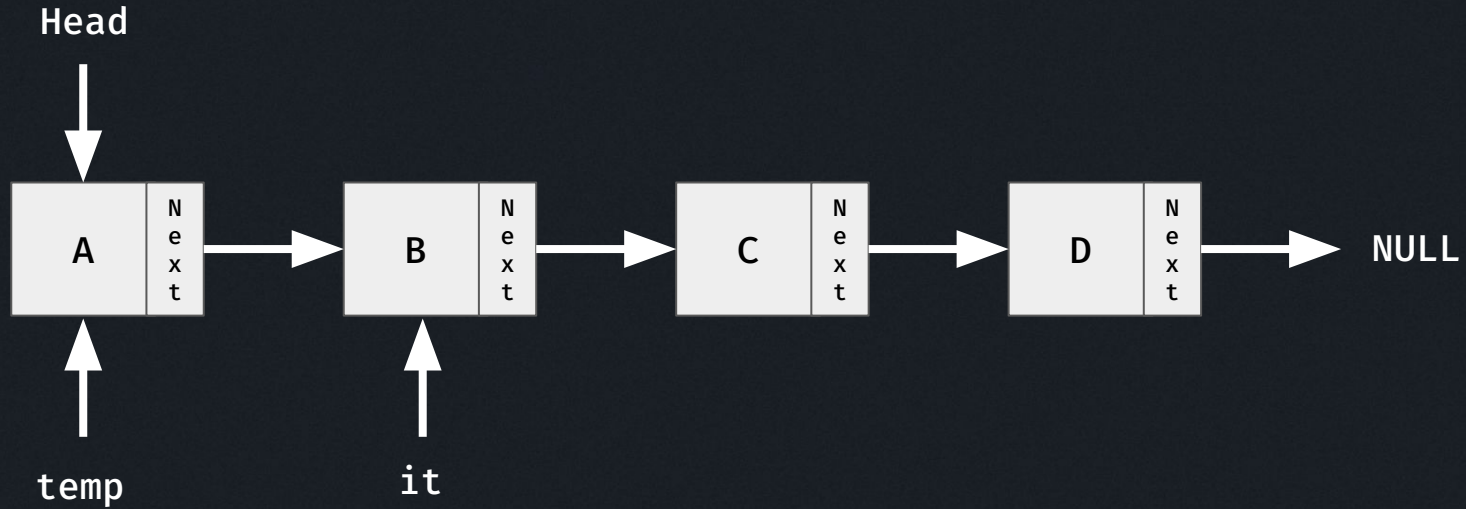
<https://leetcode.com/problems/reverse-linked-list/>



Prev → NULL

Reverse Linked List

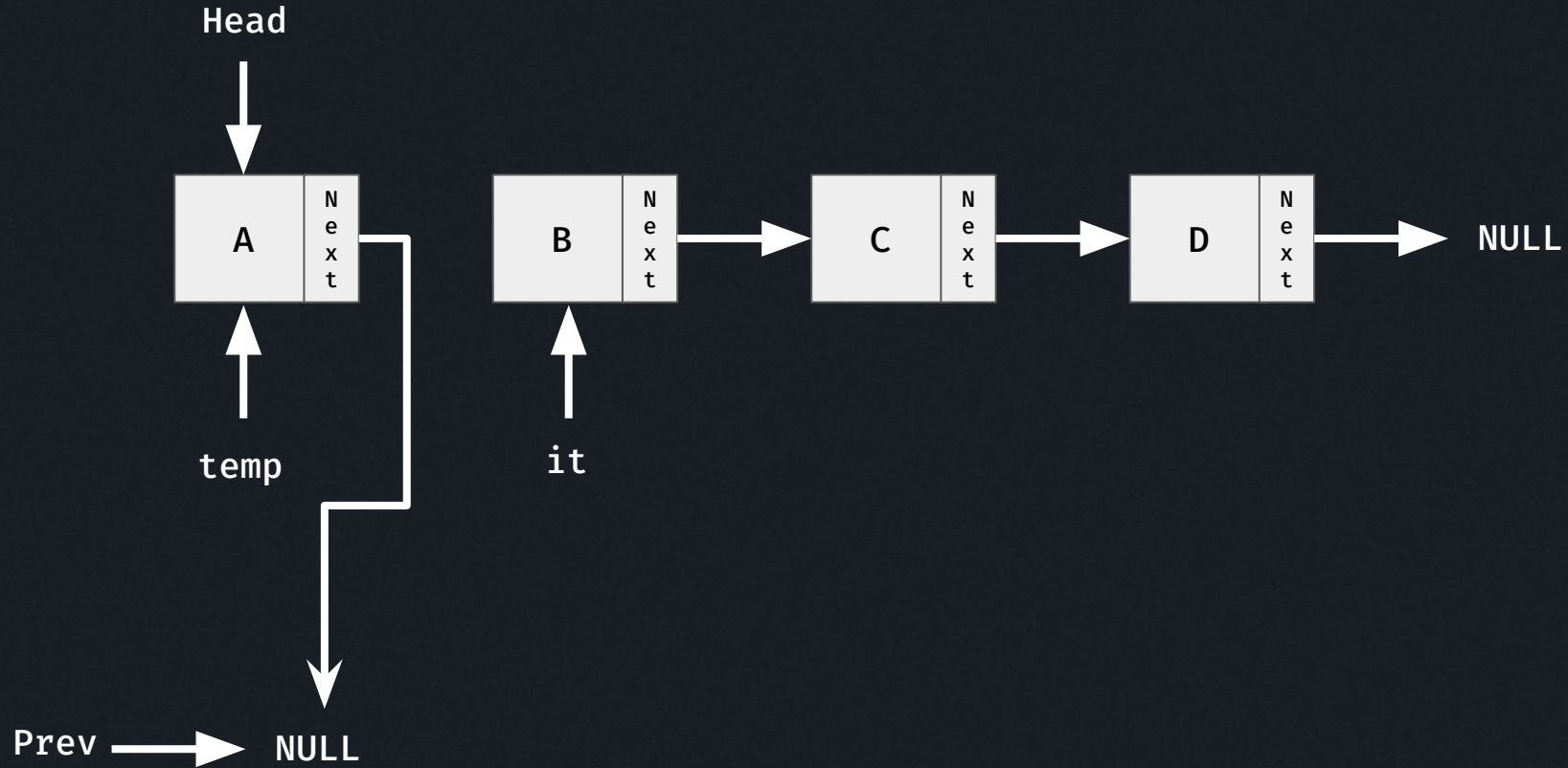
<https://leetcode.com/problems/reverse-linked-list/>



Prev \longrightarrow NULL

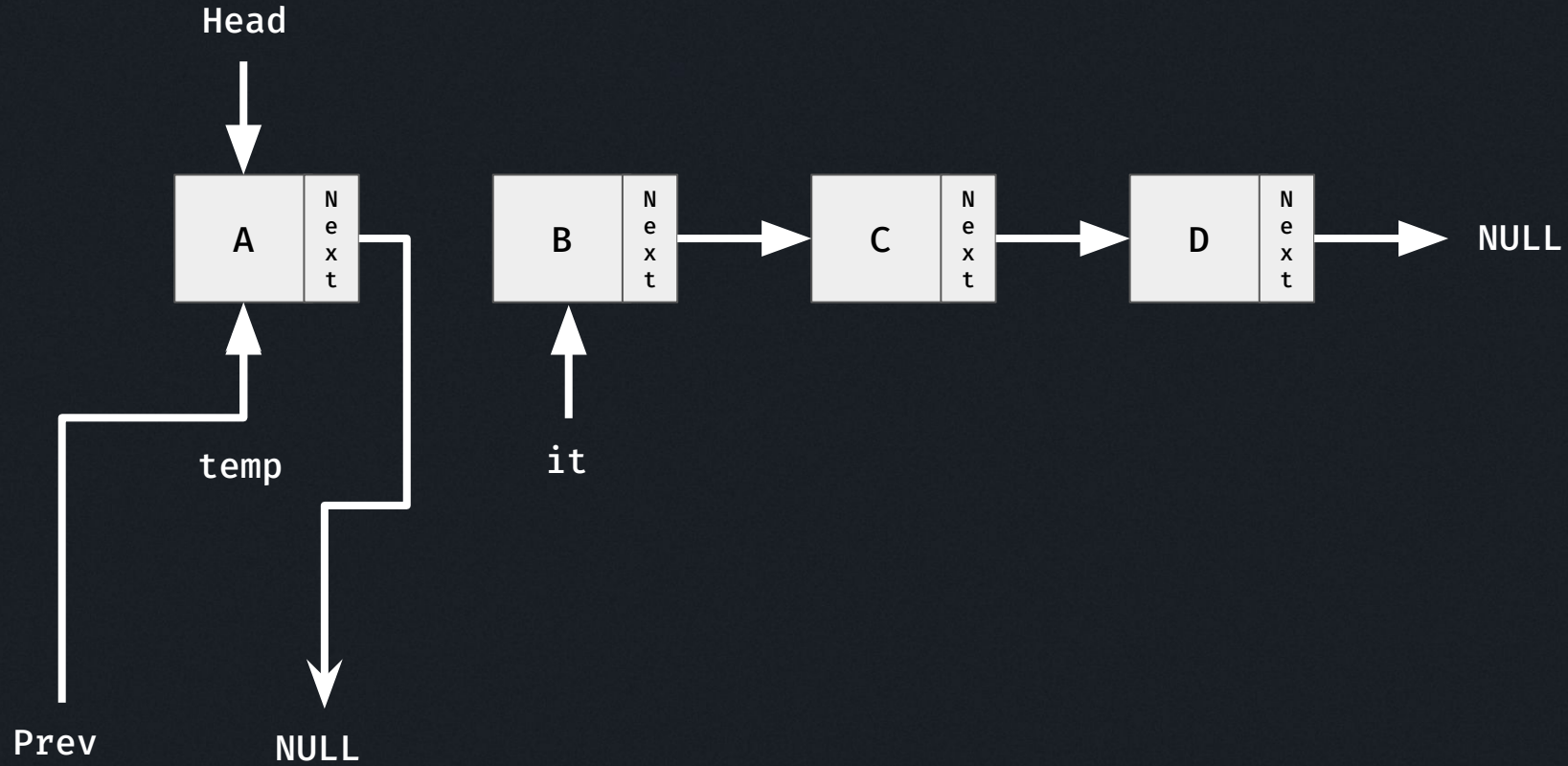
Reverse Linked List

<https://leetcode.com/problems/reverse-linked-list/>



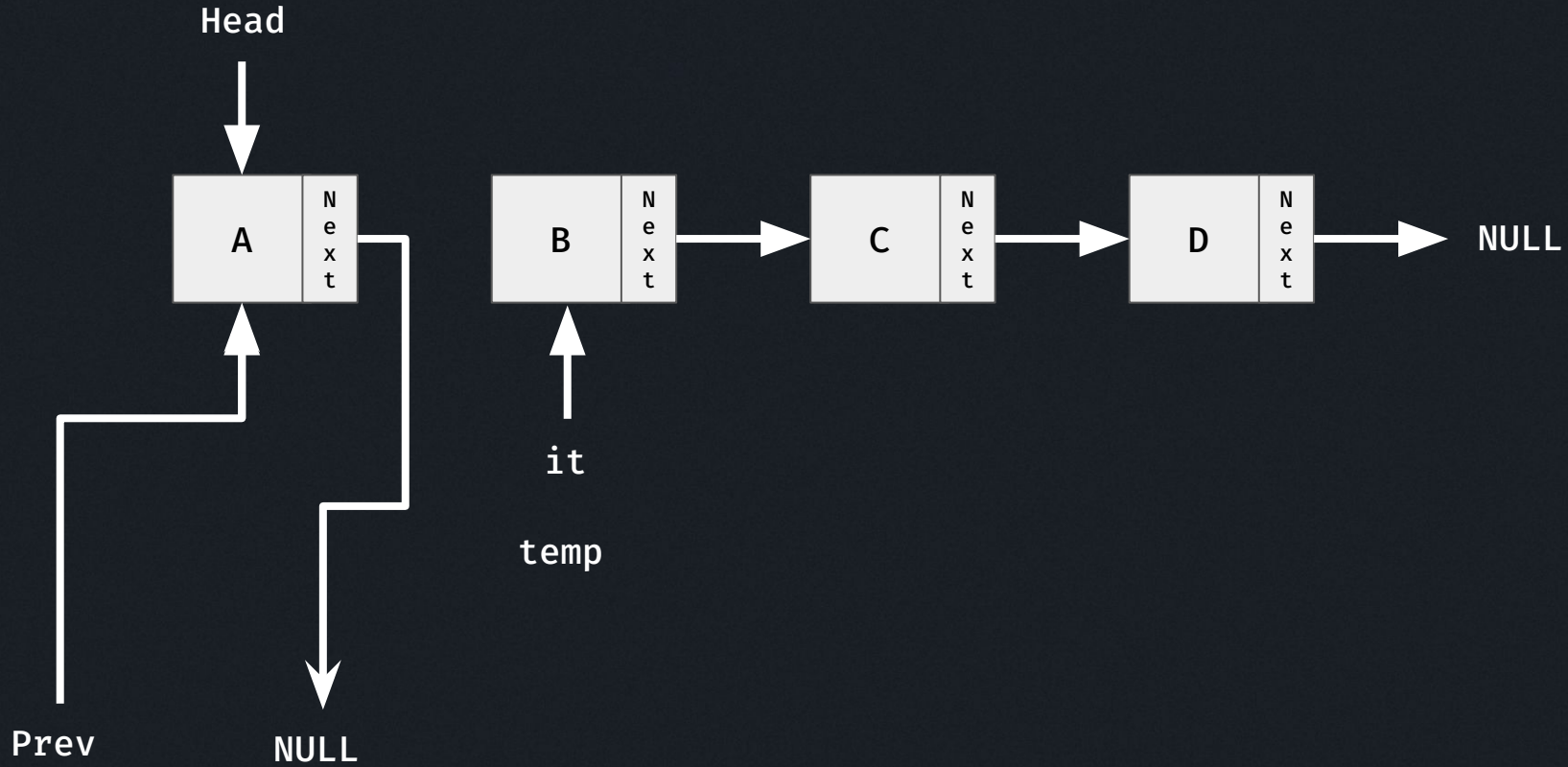
Reverse Linked List

<https://leetcode.com/problems/reverse-linked-list/>



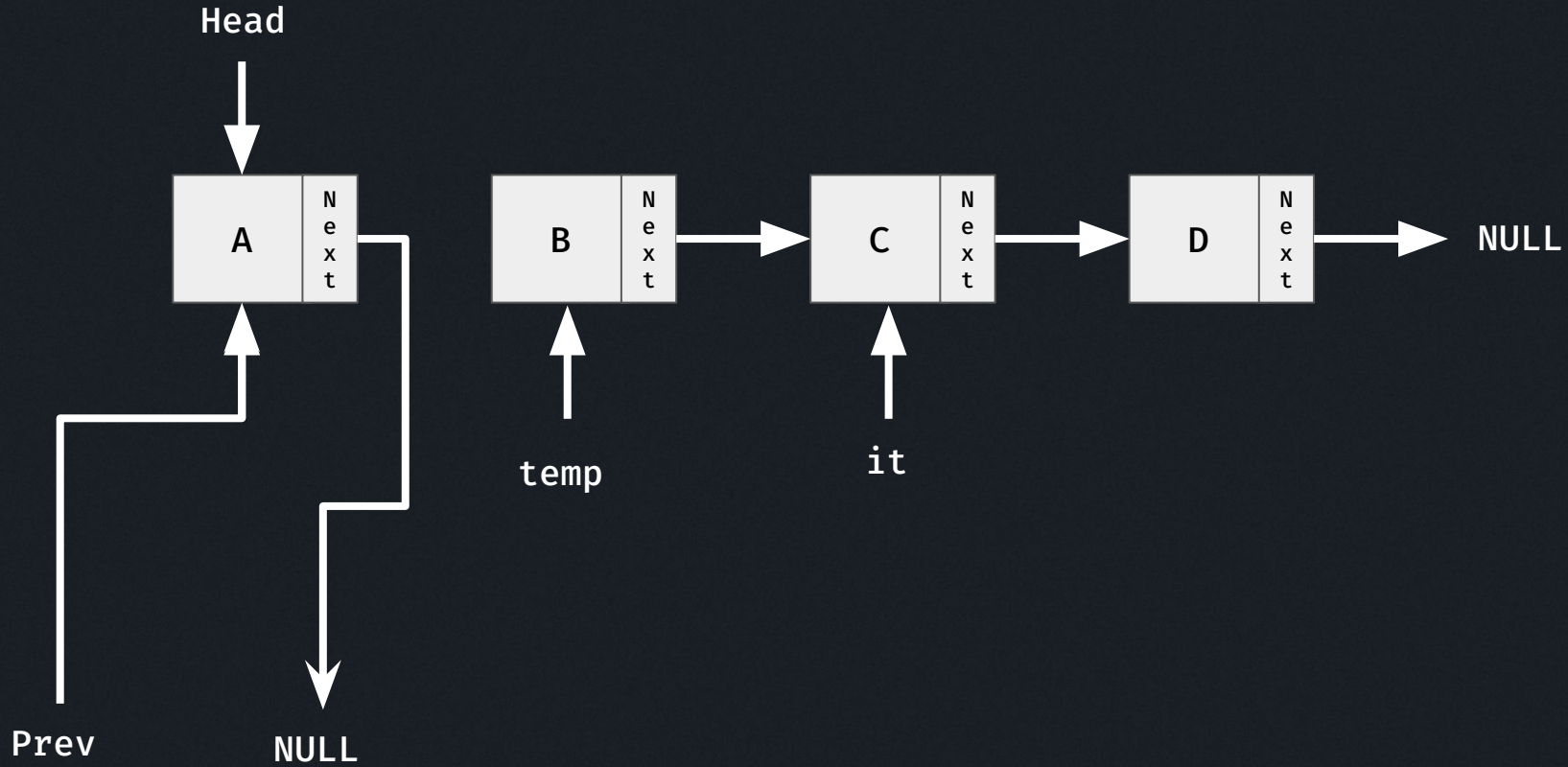
Reverse Linked List

<https://leetcode.com/problems/reverse-linked-list/>



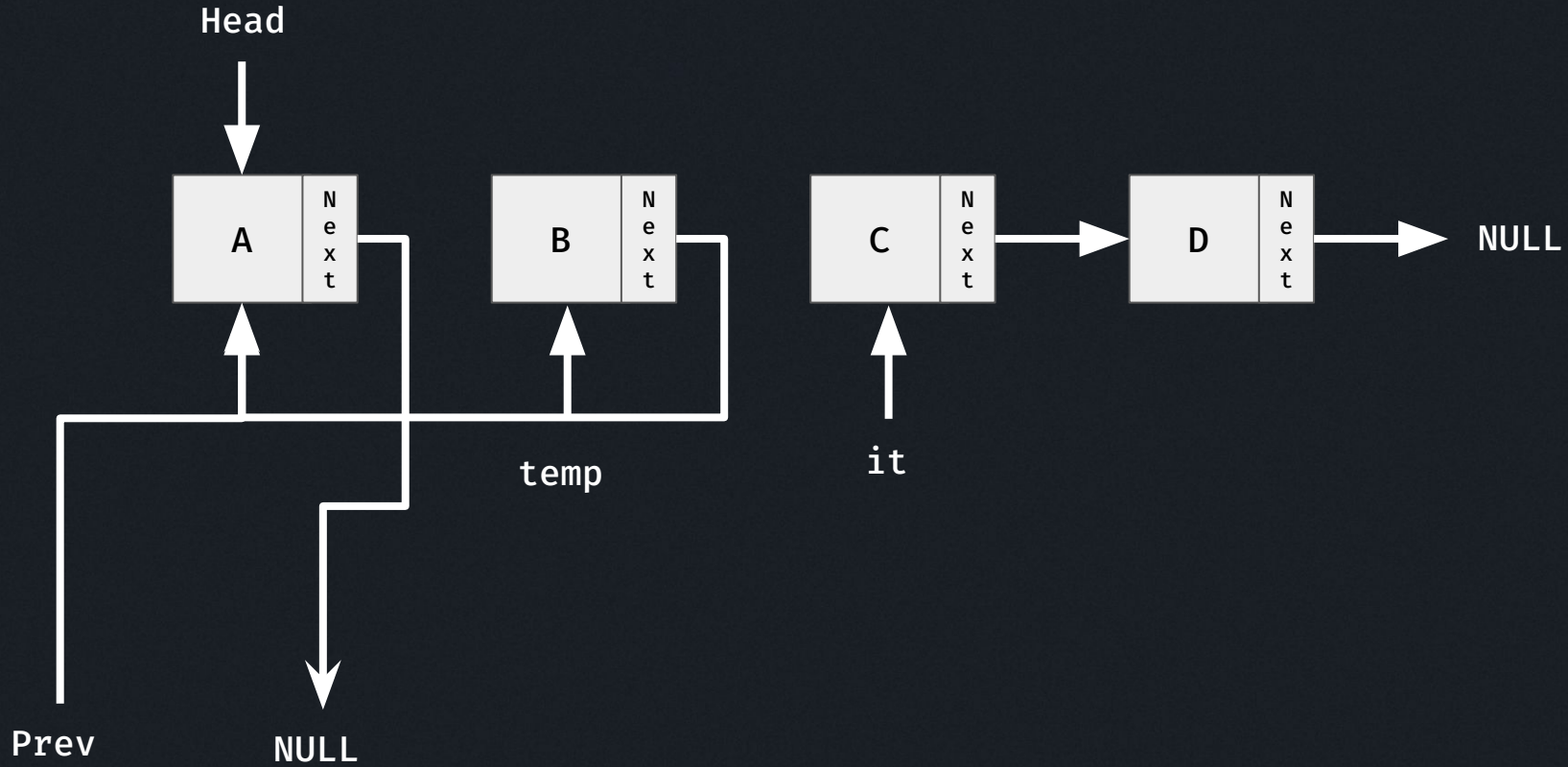
Reverse Linked List

<https://leetcode.com/problems/reverse-linked-list/>



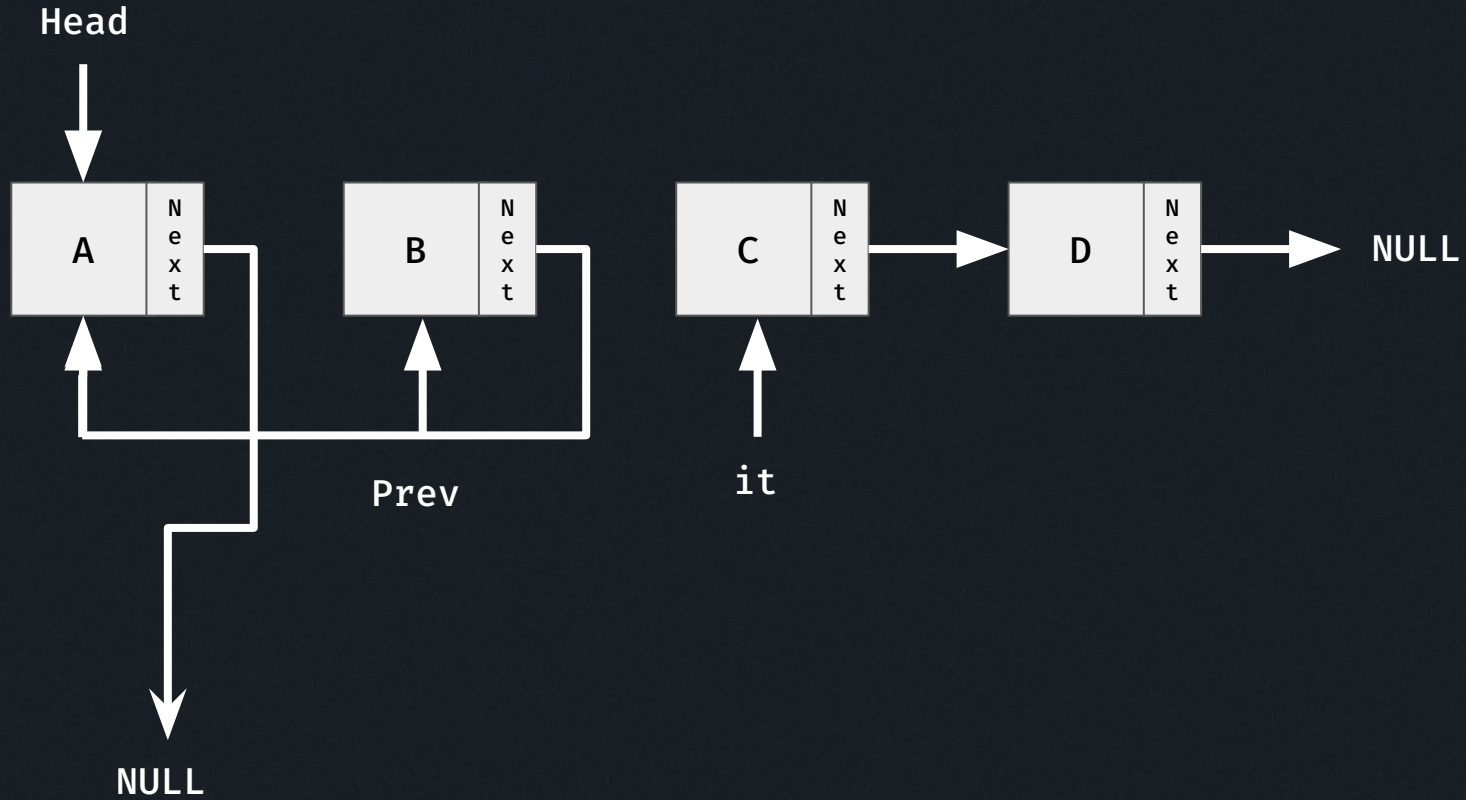
Reverse Linked List

<https://leetcode.com/problems/reverse-linked-list/>



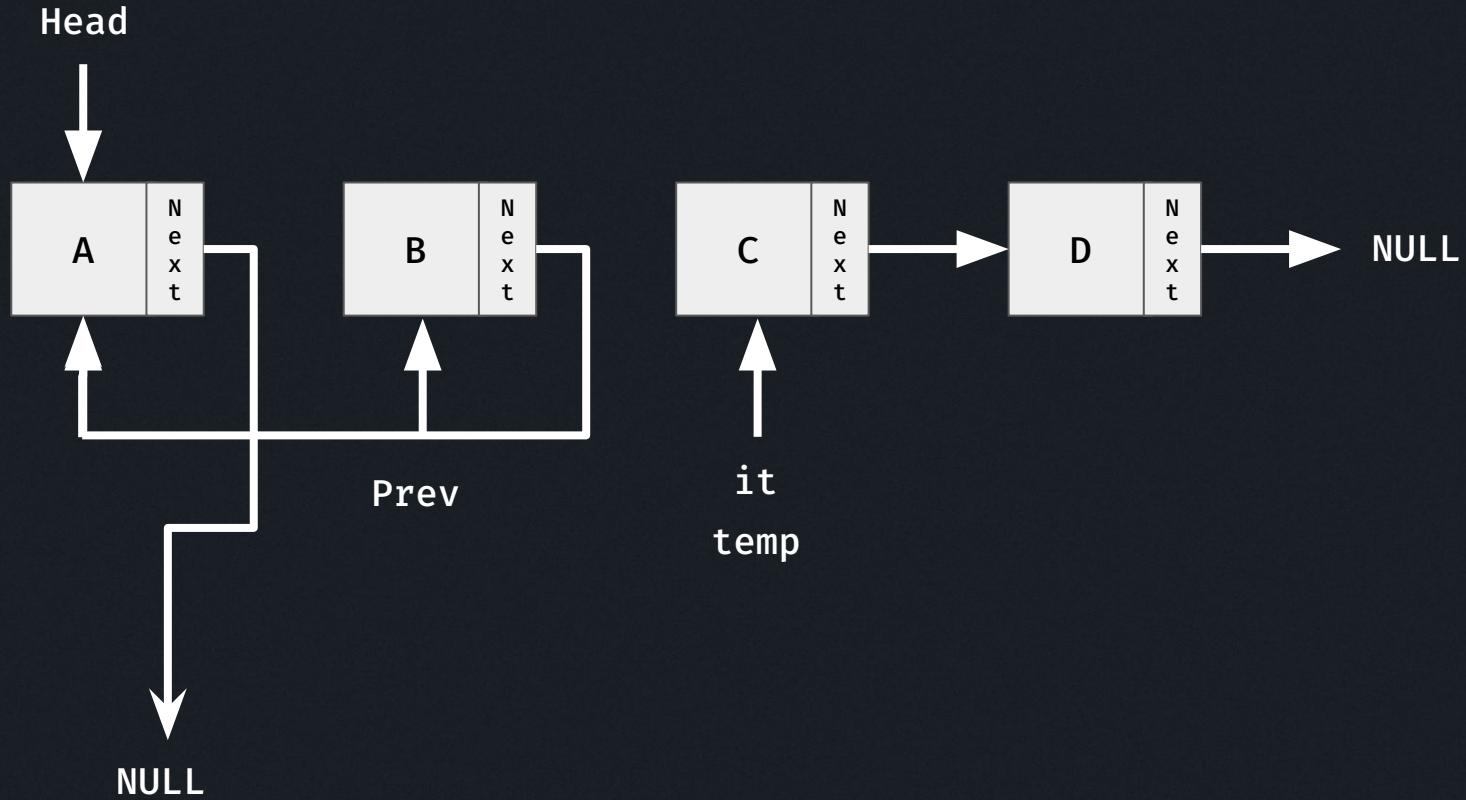
Reverse Linked List

<https://leetcode.com/problems/reverse-linked-list/>



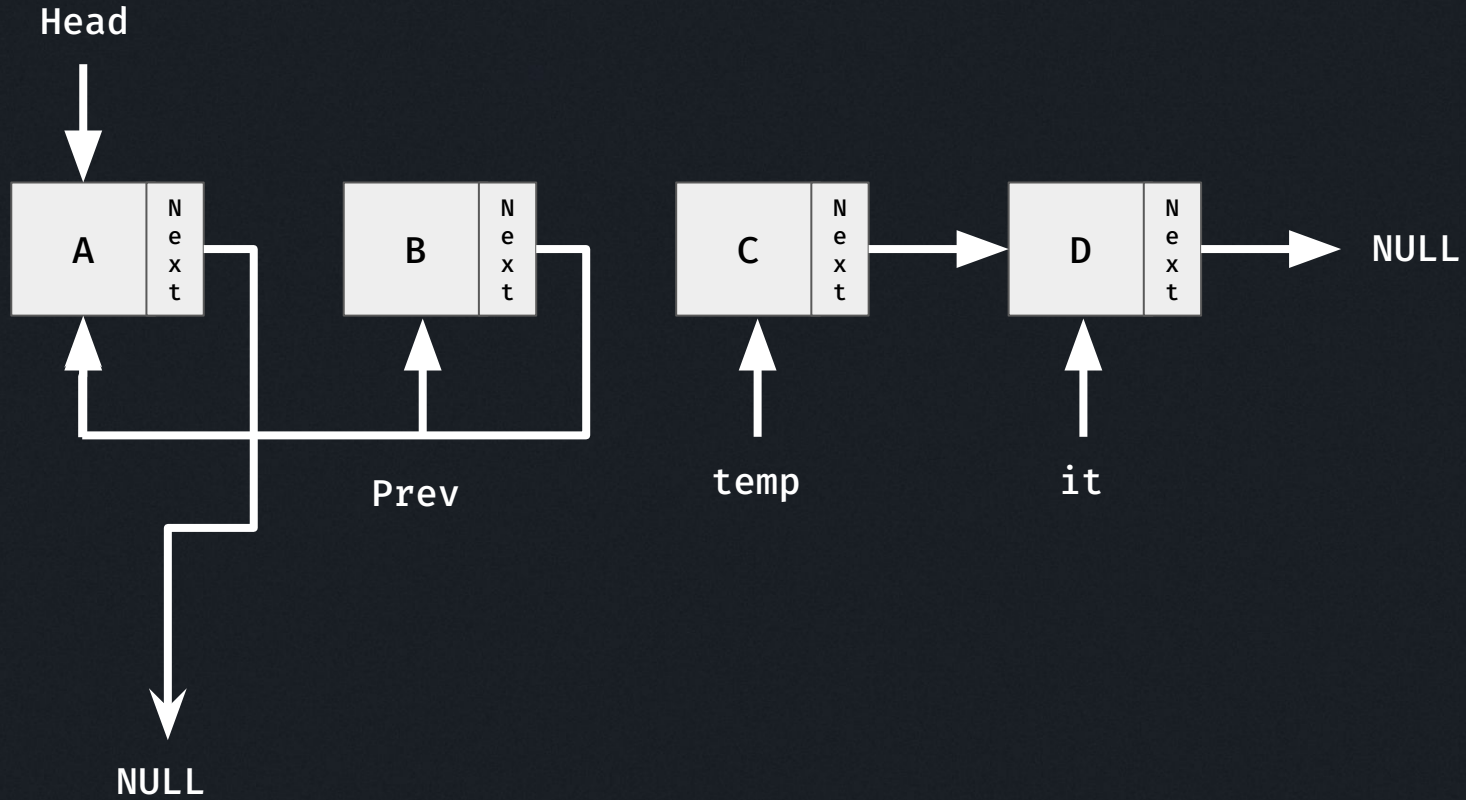
Reverse Linked List

<https://leetcode.com/problems/reverse-linked-list/>



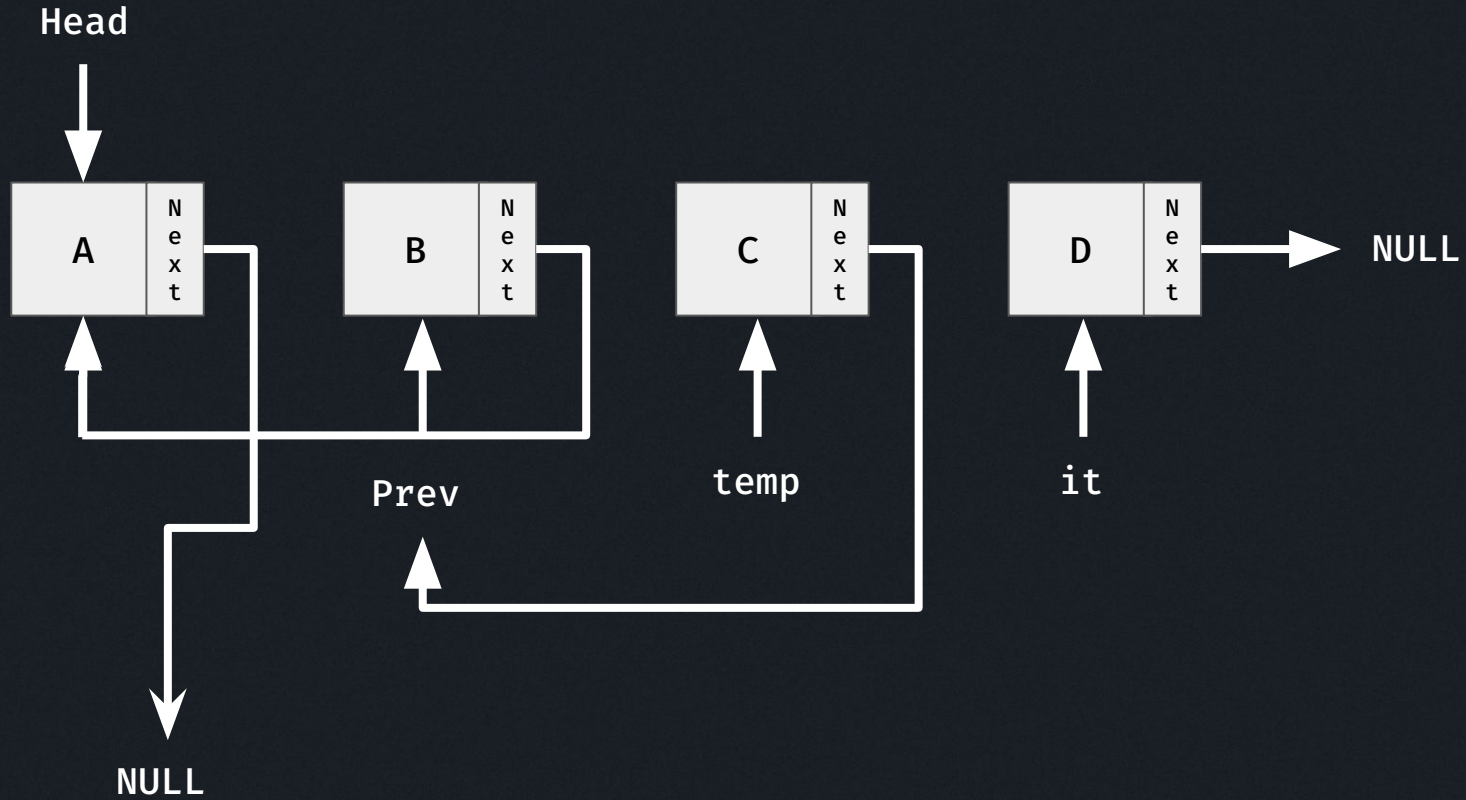
Reverse Linked List

<https://leetcode.com/problems/reverse-linked-list/>



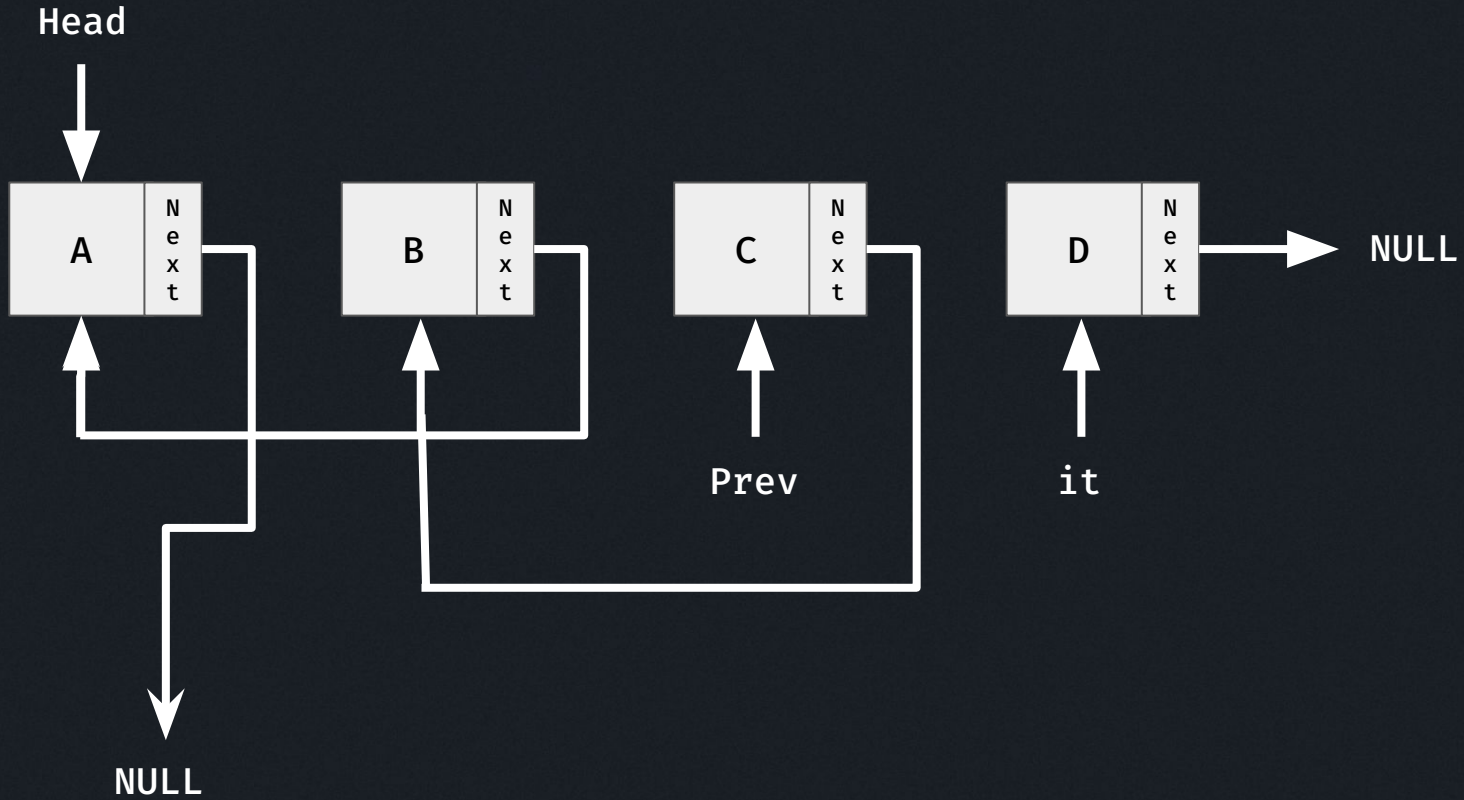
Reverse Linked List

<https://leetcode.com/problems/reverse-linked-list/>



Reverse Linked List

<https://leetcode.com/problems/reverse-linked-list/>

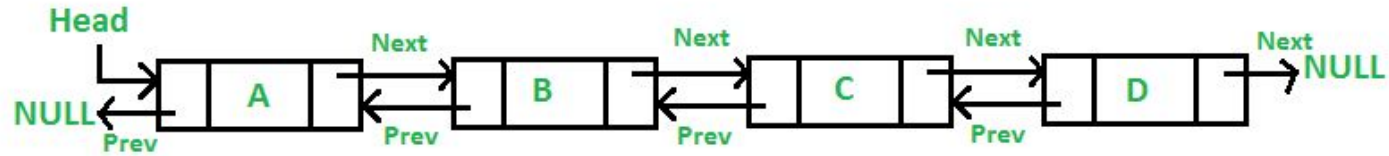


Middle of the Linked List

<https://leetcode.com/problems/middle-of-the-linked-list/>

```
ListNode ReverseList(ListNode head) {  
    ListNode it = head;  
    ListNode prev = null;  
    while (it != null) {  
        ListNode temp = it;  
        it = it.next;  
        temp.next = prev;  
        prev = temp;  
    }  
    return prev;  
}
```

Listas duplamente encadeadas



Listas Circulares



Exercícios

2

- Add Two Numbers – LeetCode
- Swap Nodes in Pairs – LeetCode
- Swapping Nodes in a Linked List – LeetCode

Exercícios

3

- [Merge Two Sorted Lists - LeetCode](#)
- [Partition List - LeetCode](#)
- [Merge k Sorted Lists - LeetCode](#)

Recommended reading

- Introduction to Algorithms - Thomas H Cormen (10.2 Linked Lists)
- How dynamic arrays work (Combination between Array + Linked List)
<https://www.geeksforgeeks.org/how-do-dynamic-arrays-work/>
- Tortoise and Hare algorithm
https://www.youtube.com/watch?v=_iB5d55tJMo

Obrigado!