Pilhas e Filas

PrepTech Google

Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

150. Evaluate Reverse Polish Notation

Medium | Topics | Companies

You are given an array of strings `tokens` that represents an arithmetic expression in a Reverse Polish Notation.

Evaluate the expression. Return *an integer that represents the value of the expression.*

Note that:

- The valid operators are `'+'`, `'-'`, `'*'`, and `'/'`.
- Each operand may be an integer or another expression.
- The division between two integers always **truncates toward zero**.
- There will not be any division by zero.
- The input represents a valid arithmetic expression in a reverse polish notation.
- The answer and all the intermediate calculations can be represented in a **32-bit** integer.

Example 1:

```
Input: tokens = ["2","1","+","3","*"]
Output: 9
Explanation: ((2 + 1) * 3) = 9
```

Example 2:

```
Input: tokens = ["4","13","5","/","+"]
Output: 6
Explanation: (4 + (13 / 5)) = 6
```

Example 3:

```
Input: tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]
Output: 22
Explanation: ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22
```

Constraints:

- `1 <= tokens.length <= 10^4`
- `tokens[i]` is either an operator: `"+"`, `"-"`, `"*"`, or `"/"`, or an integer in the range `[-200, 200]`.

Como resolver esse problema?

Pilhas

Uma pilha é uma estrutura de dados linear que segue o princípio LIFO (Last In, First Out), onde o último elemento inserido é o primeiro a ser removido.

ada

Pilhas

Operações básicas:

- push: adiciona um elemento ao topo da pilha
- pop: remove o elemento do topo da pilha
- top/peek: acessa o elemento do topo da pilha sem removê-lo

Pilhas

Implementação

```
class Stack:
    def __init__(self): # constructor

    def push(self, item):

    def pop(self):

    def top(self):
```

Pilhas

Push - Adicionando elementos na pilha

Elementos sempre são adicionados ao topo da pilha. Dizemos que elementos são “empilhados”

```
stack.push(120)
```

Pilhas

Push - Adicionando elementos na pilha

Elementos sempre são adicionados ao topo da pilha. Dizemos que elementos são “empilhados”

```
stack.push(120)
```

Pilhas

Pop - Removendo elementos da pilha

Elementos sempre são retirados do topo da pilha. Dizemos que elementos são “desempilhados”

```
stack.pop() # retorna 120
```

Pilhas

Pop - Removendo elementos da pilha

Elementos sempre são retirados do topo da pilha. Dizemos que elementos são “desempilhados”

```
stack.pop() # retorna 120
```

Pilhas

Top - Acessando o último elemento inserido

A função top (algumas implementações chamam essa operação de peek) acessa o último elemento inserido na pilha, sem removê-lo

```
stack.top() # retorna 50
```

Pilhas

Implementação

```
class Stack:
    def __init__(self): # constructor
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        return None

    def top(self):
        if not self.is_empty():
            return self.items[-1]
        return None

    def is_empty(self):
        return len(self.items) == 0
```

Pilhas

Implementação

Todas essas operações possuem complexidade O(1) de tempo

```
class Stack:
    def __init__(self): # constructor
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        return None

    def top(self):
        if not self.is_empty():
            return self.items[-1]
        return None

    def is_empty(self):
        return len(self.items) == 0
```

Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

150. Evaluate Reverse Polish Notation

Medium | Topics | Companies

You are given an array of strings `tokens` that represents an arithmetic expression in a Reverse Polish Notation.

Evaluate the expression. Return *an integer that represents the value of the expression.*

Note that:

- The valid operators are `'+'`, `'-'`, `'*'`, and `'/'`.
- Each operand may be an integer or another expression.
- The division between two integers always **truncates toward zero**.
- There will not be any division by zero.
- The input represents a valid arithmetic expression in a reverse polish notation.
- The answer and all the intermediate calculations can be represented in a **32-bit** integer.

Example 1:

```
Input: tokens = ["2","1","+","3","*"]
Output: 9
Explanation: ((2 + 1) * 3) = 9
```

Example 2:

```
Input: tokens = ["4","13","5","/","+"]
Output: 6
Explanation: (4 + (13 / 5)) = 6
```

Example 3:

```
Input: tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]
Output: 22
Explanation: ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22
```

Constraints:

- `1 <= tokens.length <= 10^4`
- `tokens[i]` is either an operator: `"+"`, `"-"`, `"*"`, or `"/"`, or an integer in the range `[-200, 200]`.

Como resolver esse problema utilizando pilha?

Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

150. Evaluate Reverse Polish Notation

Medium | Topics | Companies

You are given an array of strings `tokens` that represents an arithmetic expression in a Reverse Polish Notation.

Evaluate the expression. Return *an integer that represents the value of the expression.*

Note that:

- The valid operators are `'+'`, `'-'`, `'*'`, and `'/'`.
- Each operand may be an integer or another expression.
- The division between two integers always **truncates toward zero**.
- There will not be any division by zero.
- The input represents a valid arithmetic expression in a reverse polish notation.
- The answer and all the intermediate calculations can be represented in a **32-bit** integer.

Example 1:

```
Input: tokens = ["2","1","+","3","*"]
Output: 9
Explanation: ((2 + 1) * 3) = 9
```

Example 2:

```
Input: tokens = ["4","13","5","/","+"]
Output: 6
Explanation: (4 + (13 / 5)) = 6
```

Example 3:

```
Input: tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]
Output: 22
Explanation: ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22
```

Constraints:

- `1 <= tokens.length <= 10^4`
- `tokens[i]` is either an operator: `"+"`, `"-"`, `"*"`, or `"/"`, or an integer in the range `[-200, 200]`.

Estratégia: percorrer o array de tokens empilhando os operandos.

Quando encontrar um operador, retira os dois últimos operandos da pilha e realiza a operação, empilhando o resultado.

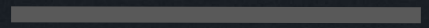
Como a entrada é garantida ser uma expressão válida, ao final das operações a pilha conterá um único valor, que é a avaliação da expressão de entrada.

ada

Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```



ada

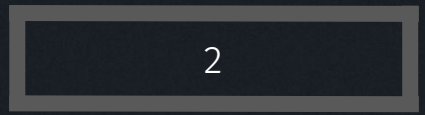
Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

token "2" é um operando, então empilhamos

```
stack.push(2)
```



ada

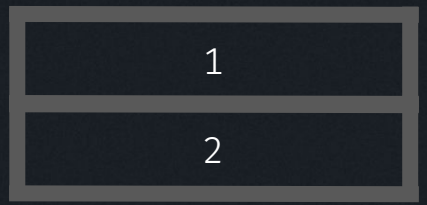
Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

token "1" é um operando, então empilhamos

```
stack.push(1)
```



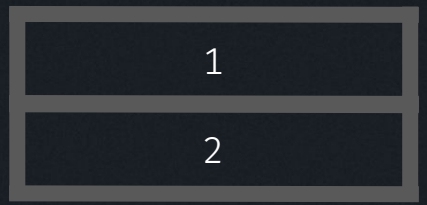
ada

Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

token "+" é um operador, então desempilhamos os últimos 2 operandos e realizamos a operação de soma, empilhando o resultado



ada

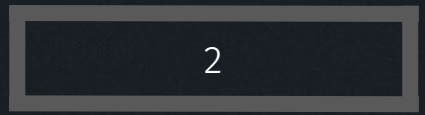
Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

token "+" é um operador, então desempilhamos os últimos 2 operandos e realizamos a operação de soma, empilhando o resultado

```
a = stack.pop() # 1
```



ada

Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

token "+" é um operador, então desempilhamos os últimos 2 operandos e realizamos a operação de soma, empilhando o resultado

```
a = stack.pop() # 1
b = stack.pop() # 2
```

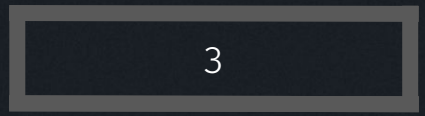
Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

token "+" é um operador, então desempilhamos os últimos 2 operandos e realizamos a operação de soma, empilhando o resultado

```
a = stack.pop() # 1
b = stack.pop() # 2
result = b + a # 3
stack.push (result)
```



ada

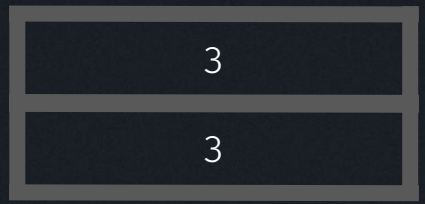
Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

token "3" é um operando, então empilhamos

```
stack.push(3)
```



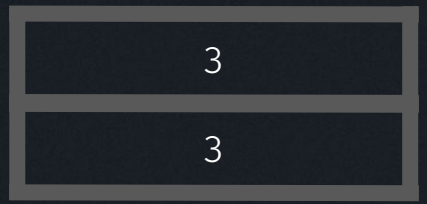
ada

Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

token "*" é um operador, então desempilhamos os últimos 2 operandos e realizamos a operação de multiplicação, empilhando o resultado



ada

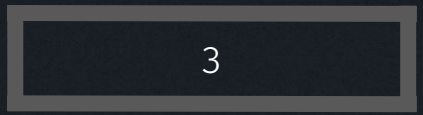
Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

token "*" é um operador, então desempilhamos os últimos 2 operandos e realizamos a operação de multiplicação, empilhando o resultado

```
a = stack.pop() # 3
```



ada

Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

token "*" é um operador, então desempilhamos os últimos 2 operandos e realizamos a operação de multiplicação, empilhando o resultado

```
a = stack.pop() # 3
b = stack.pop() # 3
```

ada

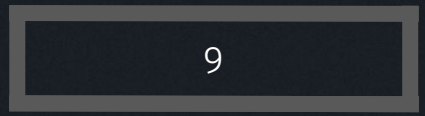
Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

token "*" é um operador, então desempilhamos os últimos 2 operandos e realizamos a operação de multiplicação, empilhando o resultado

```
a = stack.pop() # 3
b = stack.pop() # 3
result = b * a # 9
stack.push (result)
```



ada

Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

Terminamos de percorrer o array `tokens`, a avaliação da expressão é o valor no topo da pilha

9

ada

Evaluate Reverse Polish Notation

https://leetcode.com/problems/evaluate-reverse-polish-notation/

```
tokens = ["2","1","+","3","*"]
```

Terminamos de percorrer o array `tokens`, a avaliação da expressão é o valor no topo da pilha

```
return stack.pop() # 9
```

Remove All Adjacent Duplicates In String

https://leetcode.com/problems/remove-all-adjacent-duplicates-in-string/description/

1047. Remove All Adjacent Duplicates In String

Easy | Topics | Companies | Hint

You are given a string `s` consisting of lowercase English letters. A **duplicate removal** consists of choosing two **adjacent** and **equal** letters and removing them.

We repeatedly make **duplicate removals** on `s` until we no longer can.

Return *the final string after all such duplicate removals have been made*. It can be proven that the answer is **unique**.

Example 1:

```
Input: s = "abbaca"
Output: "ca"
Explanation:
For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal, and this is the only possible move.
The result of this move is that the string is "aaca", of which only "aa" is possible, so the final string is "ca".
```

Example 2:

```
Input: s = "azxxzy"
Output: "ay"
```

Constraints:

- `1 <= s.length <= 10^5`
- `s` consists of lowercase English letters.

Remove All Adjacent Duplicates In String

https://leetcode.com/problems/remove-all-adjacent-duplicates-in-string/description/

Solução usando pilha: precisamos iterar caracter a caracter da string de entrada, verificando se o caracter atual é igual ao que está no topo, se for igual, descartar e remover o topo da pilha.

Após passar por todos os caracteres da entrada, a string formada pelos caracteres da pilha, será a string resultante.

ada

Remove All Adjacent Duplicates In String

https://leetcode.com/problems/remove-all-adjacent-duplicates-in-string/description/

C++ Solution:

```cpp
    string removeDuplicates(string s) {
        string output;
        for (const char& c: s) {
            if (output.empty()) {
                output.push_back(c);
                continue;
            }

            if (output.back() == c) {
                output.pop_back();
            }
            else {
                output.push_back(c);
            }
        }

        return output;
    }
```

ada

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

2073. Time Needed to Buy Tickets

Solved

Easy | Topics | Companies | Hint

There are `n` people in a line queuing to buy tickets, where the `0th` person is at the **front** of the line and the `(n - 1)th` person is at the **back** of the line.

You are given a **0-indexed** integer array `tickets` of length `n` where the number of tickets that the `ith` person would like to buy is `tickets[i]`.

Each person takes **exactly 1 second** to buy a ticket. A person can only buy **1 ticket at a time** and has to go back to **the end** of the line (which happens **instantaneously**) in order to buy more tickets. If a person does not have any tickets left to buy, the person will **leave** the line.

Return *the **time taken** for the person at position `k` **(0-indexed)** to finish buying tickets.*

Example 1:

> **Input:** tickets = [2,3,2], k = 2
> **Output:** 6
> **Explanation:**
> - In the first pass, everyone in the line buys a ticket and the line becomes [1, 2, 1].
> - In the second pass, everyone in the line buys a ticket and the line becomes [0, 1, 0].
> The person at position 2 has successfully bought 2 tickets and it took 3 + 3 = 6 seconds.

Example 2:

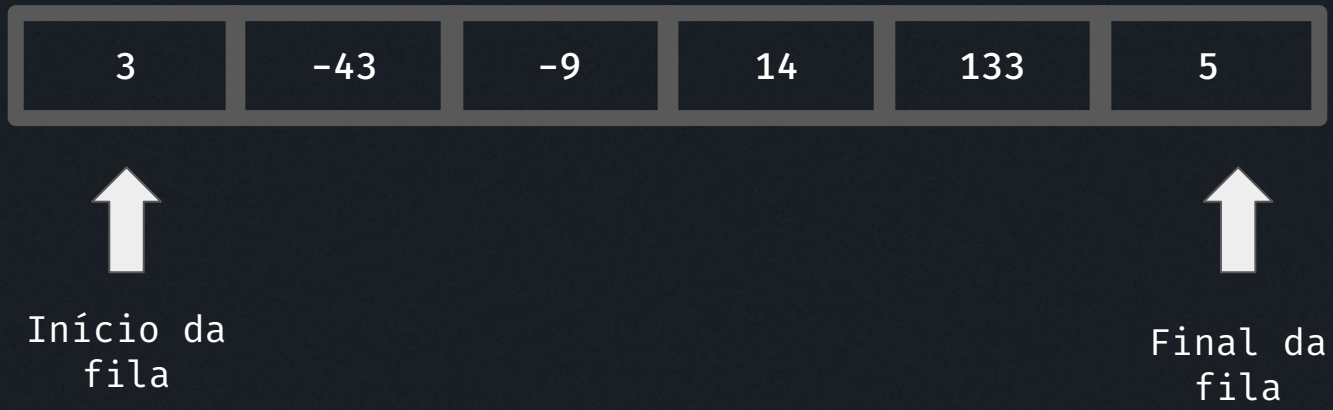
> **Input:** tickets = [5,1,1,1], k = 0
> **Output:** 8
> **Explanation:**
> - In the first pass, everyone in the line buys a ticket and the line becomes [4, 0, 0, 0].
> - In the next 4 passes, only the person in position 0 is buying tickets.
> The person at position 0 has successfully bought 5 tickets and it took 4 + 1 + 1 + 1 + 1 = 8 seconds.

Como resolver esse problema?

ada

Filas

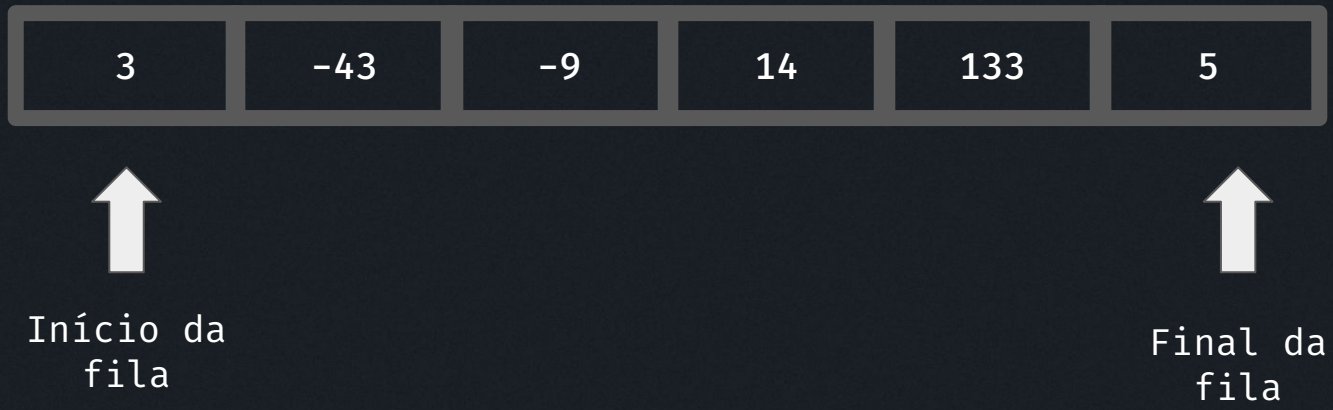
Uma fila é uma estrutura de dados linear que segue o princípio FIFO (First In, First Out), onde o primeiro elemento inserido é o primeiro a ser removido. Ou seja, elementos são removidos na mesma ordem de inserção.



ada

Operações básicas:

- enqueue: adiciona um elemento ao final da fila
- dequeue: remove o elemento do início da fila
- front/peek: acessa o elemento do início da fila sem removê-lo



Filas

Implementação

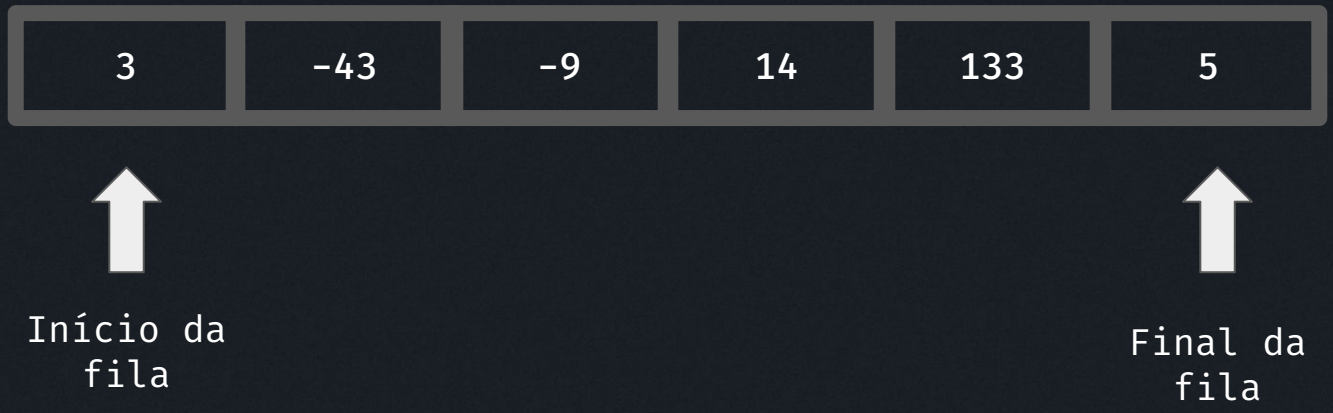
```
class Queue:
    def __init__(self): # constructor

    def enqueue(self, item):

    def dequeue(self):

    def front(self):

    def is_empty(self):
```

Filas

Enqueue - Adicionando elementos na fila

Elementos sempre são adicionados ao final da fila. Dizemos que elementos são “enfileirados”

```
queue.enqueue(120)
```

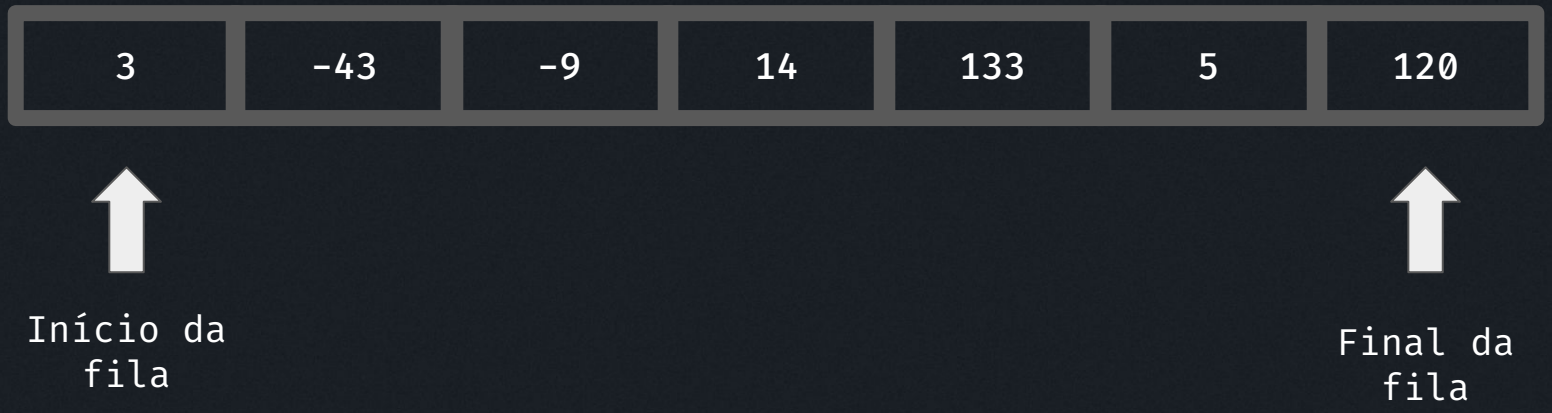


Filas

Enqueue - Adicionando elementos na fila

Elementos sempre são adicionados ao final da fila. Dizemos que elementos são “enfileirados”

```
queue.enqueue(120)
```

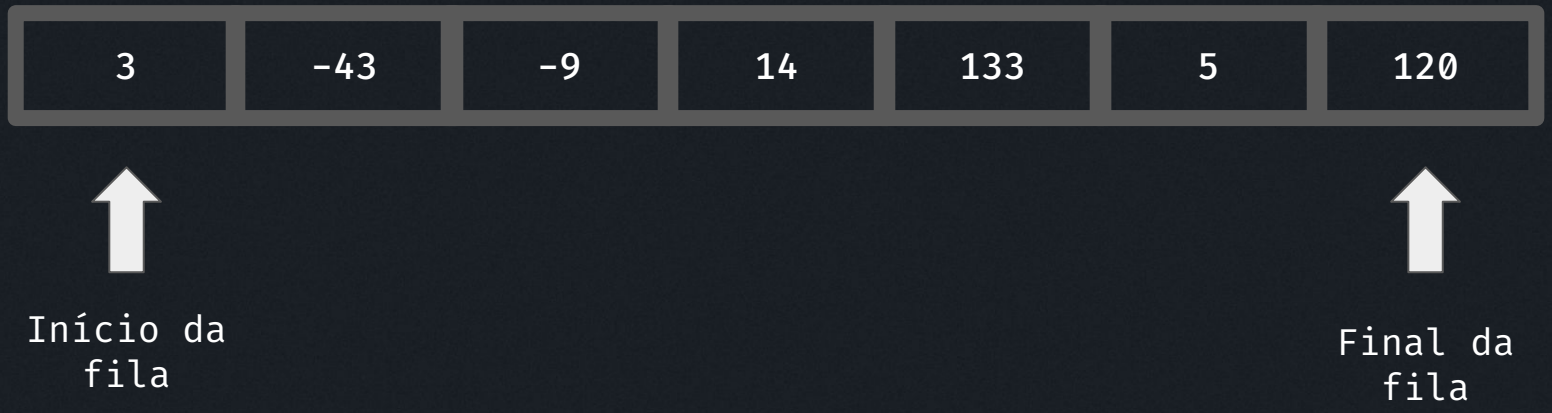


Filas

Dequeue- Removendo elementos da fila

Elementos sempre são removidos do início da fila. Dizemos que elementos são “desenfileirados”

```
queue.dequeue() # 3
```

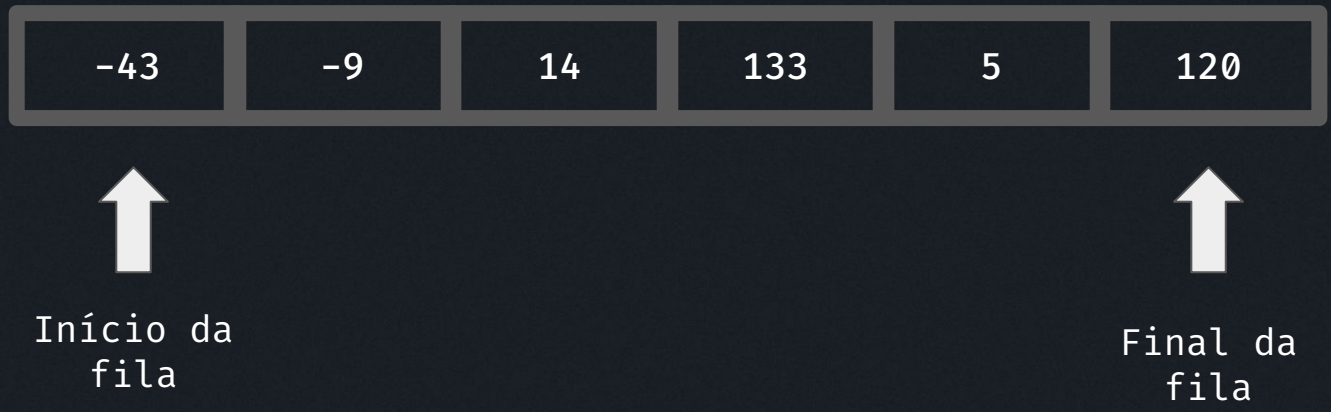


Filas

Dequeue- Removendo elementos da fila

Elementos sempre são removidos do início da fila. Dizemos que elementos são “desenfileirados”

```
queue.dequeue() # 3
```

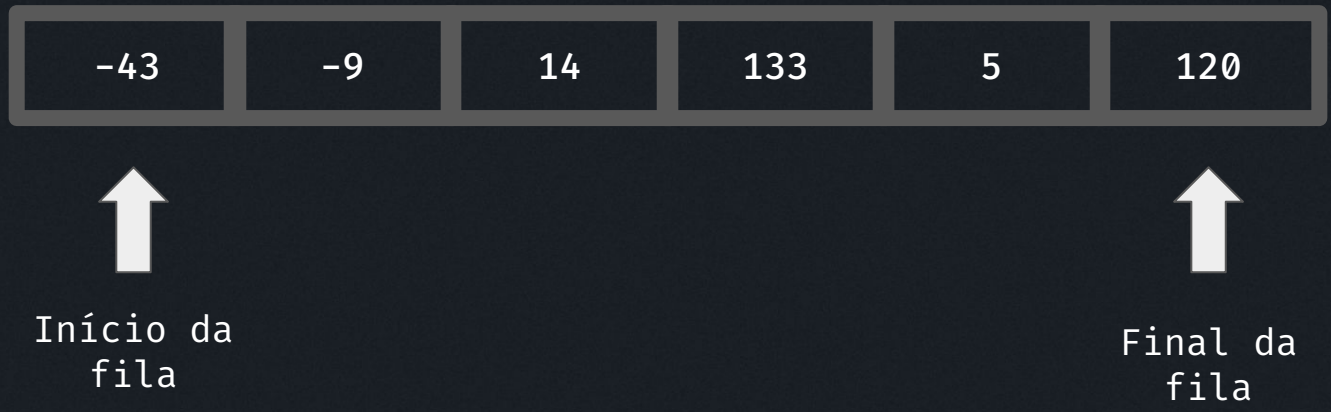


Filas

Front - Acessando o elemento do início da fila

A função front (algumas implementações chamam essa função é peek) acessa o elemento do início da fila, sem removê-lo.

```
queue.front() # -43
```



Filas

Implementação com array

```python
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)
        return None

    def front(self):
        if not self.is_empty():
            return self.items[0]
        return None

    def is_empty(self):
        return len(self.items) == 0
```

Filas

Implementação com array

Qual é a complexidade dessa implementação?

```python
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)
        return None

    def front(self):
        if not self.is_empty():
            return self.items[0]
        return None

    def is_empty(self):
        return len(self.items) == 0
```

Filas

Implementação com array

Adicionar um elemento ao final do array: O(1)

```
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)
        return None

    def front(self):
        if not self.is_empty():
            return self.items[0]
        return None

    def is_empty(self):
        return len(self.items) == 0
```

Filas

Implementação com array

Verificar se o array está vazio: O(1)

```
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)
        return None

    def front(self):
        if not self.is_empty():
            return self.items[0]
        return None

    def is_empty(self):
        return len(self.items) == 0
```

Filas

Implementação com array

Retornar o primeiro elemento do array: O(1)

```
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)
        return None

    def front(self):
        if not self.is_empty():
            return self.items[0]
        return None

    def is_empty(self):
        return len(self.items) == 0
```

Filas

Implementação com array

Remover do início do array: O(N)

```
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)
        return None

    def front(self):
        if not self.is_empty():
            return self.items[0]
        return None

    def is_empty(self):
        return len(self.items) == 0
```

Filas

Implementação com lista encadeada

```
class Node:
    def __init__(self, data=None): # constructor
        self.data = data
        self.next = None


class Queue:

    def __init__(self): # constructor

        self.front = None

        self.rear = None


    def is_empty(self):


    def enqueue(self, item):


    def dequeue(self):


    def front(self):
```

Filas

Implementação com lista encadeada

Quando a fila estiver vazia, `self.front` e `self.rear` são iguais a `None`

```
def is_empty(self):
    return self.front == None
```

Filas

Implementação com lista encadeada

Para inserir na fila, temos 2 casos:

- fila vazia: criamos um novo nó e atualizamos a referência do início (front) e final da fila (rear)
- fila não-vazia: criamos um novo nó e atualizamos apenas referência do final da fila (rear)

```
def enqueue(self, item):
    new_node = Node(item)
    if self.rear is None:
        self.front = self.rear = new_node
    else:
        self.rear.next = new_node
        self.rear = new_node
```

Filas

Implementação com lista encadeada

Para retirar do início da fila, basta atualizar a referência de início da fila para o próximo elemento. Caso a fila tenha apenas 1 elemento, devemos também atualizar a referência do final da fila.

```
def dequeue(self):
    if self.is_empty():
        return None
    temp = self.front
    self.front = temp.next
    if self.front is None:
        self.rear = None
    return temp.data
```

Filas

Implementação com lista encadeada

Para acessar o início da fila basta retornar o conteúdo da referência `front`.

```
def front(self):
    if self.is_empty():
        return None
    return self.front.data
```

Todas as operações de uma fila implementadas usando lista encadeada tem complexidade O(1) de tempo.

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

2073. Time Needed to Buy Tickets

Solved

Easy | Topics | Companies | Hint

There are `n` people in a line queuing to buy tickets, where the `0th` person is at the **front** of the line and the `(n - 1)th` person is at the **back** of the line.

You are given a **0-indexed** integer array `tickets` of length `n` where the number of tickets that the `ith` person would like to buy is `tickets[i]`.

Each person takes **exactly 1 second** to buy a ticket. A person can only buy **1 ticket at a time** and has to go back to **the end** of the line (which happens **instantaneously**) in order to buy more tickets. If a person does not have any tickets left to buy, the person will **leave** the line.

Return *the **time taken** for the person at position `k` **(0-indexed)** to finish buying tickets.*

Example 1:

> **Input:** tickets = [2,3,2], k = 2
> **Output:** 6
> **Explanation:**
> - In the first pass, everyone in the line buys a ticket and the line becomes [1, 2, 1].
> - In the second pass, everyone in the line buys a ticket and the line becomes [0, 1, 0].
> The person at position 2 has successfully bought 2 tickets and it took 3 + 3 = 6 seconds.

Example 2:

> **Input:** tickets = [5,1,1,1], k = 0
> **Output:** 8
> **Explanation:**
> - In the first pass, everyone in the line buys a ticket and the line becomes [4, 0, 0, 0].
> - In the next 4 passes, only the person in position 0 is buying tickets.
> The person at position 0 has successfully bought 5 tickets and it took 4 + 1 + 1 + 1 + 1 = 8 seconds.

Como resolver esse problema utilizando fila?

ada

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

2073. Time Needed to Buy Tickets

Solved

Easy | Topics | Companies | Hint

There are `n` people in a line queuing to buy tickets, where the `0th` person is at the **front** of the line and the `(n - 1)th` person is at the **back** of the line.

You are given a **0-indexed** integer array `tickets` of length `n` where the number of tickets that the `ith` person would like to buy is `tickets[i]`.

Each person takes **exactly 1 second** to buy a ticket. A person can only buy **1 ticket at a time** and has to go back to **the end** of the line (which happens **instantaneously**) in order to buy more tickets. If a person does not have any tickets left to buy, the person will **leave** the line.

Return *the **time taken** for the person at position `k` **(0-indexed)** to finish buying tickets.*

Example 1:

> **Input:** tickets = [2,3,2], k = 2
> **Output:** 6
> **Explanation:**
> - In the first pass, everyone in the line buys a ticket and the line becomes [1, 2, 1].
> - In the second pass, everyone in the line buys a ticket and the line becomes [0, 1, 0].
> The person at position 2 has successfully bought 2 tickets and it took 3 + 3 = 6 seconds.

Example 2:

> **Input:** tickets = [5,1,1,1], k = 0
> **Output:** 8
> **Explanation:**
> - In the first pass, everyone in the line buys a ticket and the line becomes [4, 0, 0, 0].
> - In the next 4 passes, only the person in position 0 is buying tickets.
> The person at position 0 has successfully bought 5 tickets and it took 4 + 1 + 1 + 1 + 1 = 8 seconds.

Estratégia: criar uma fila em que os elementos armazenam o índice e a quantidade de tickets restantes para comprar.

Enquanto o elemento de índice k ainda tiver tickets restantes para comprar, retiramos o primeiro elemento da fila, decrementamos a quantidade de tickets restantes para comprar, e colocamos de volta no final da fila caso necessário.

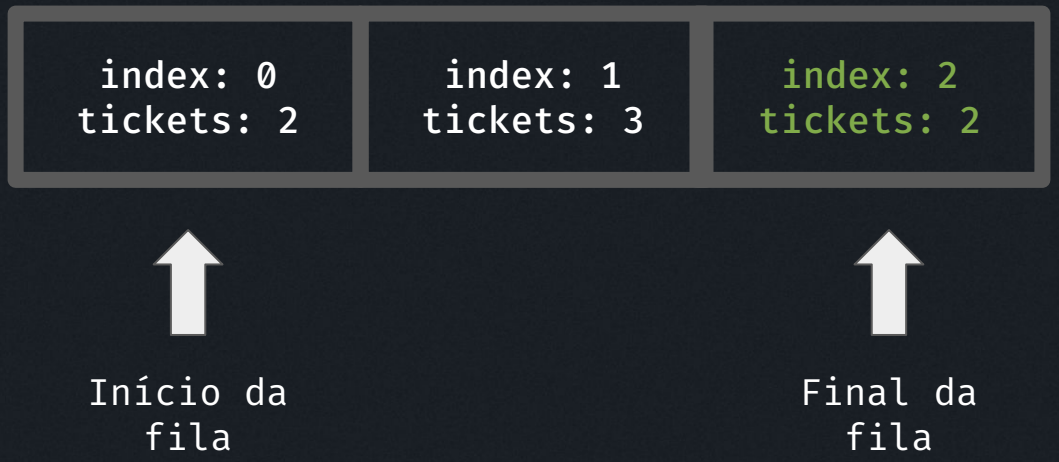
A cada iteração do loop, somamos 1 unidade de tempo.

ada

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2


time = 0
```



ada

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2
```

Retiramos o primeiro elemento da fila e incrementamos o valor de `time`

```
time = 1
```

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2
```

Decrementamos o valor de `tickets` e colocamos de volta na fila se for maior que 0

```
time = 1
```

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2
```

Decrementamos o valor de `tickets` e colocamos de volta na fila se for maior que 0

```
time = 1
```

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2
```

```
time = 2
```

Retiramos o primeiro elemento da fila e incrementamos o valor de `time`

ada

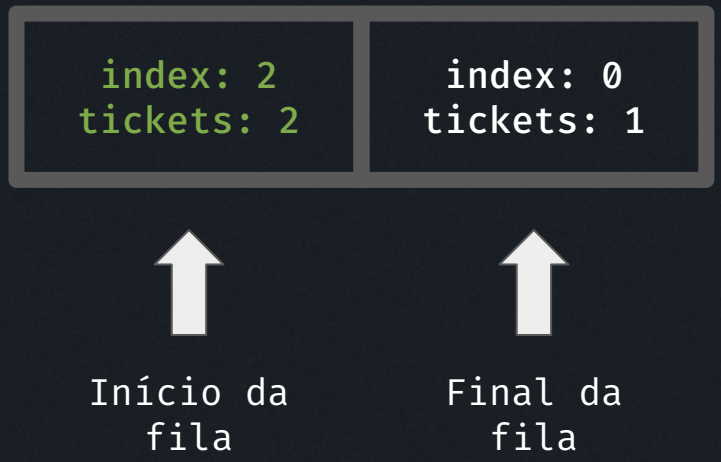
Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2
```

```
time = 2
```

Decrementamos o valor de `tickets` e colocamos de volta na fila se for maior que 0



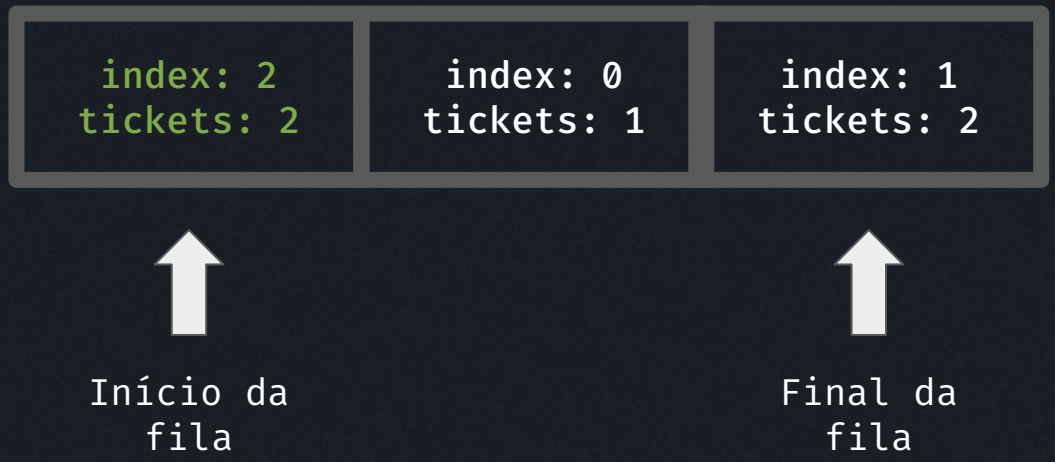
Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2
```

```
time = 2
```

Decrementamos o valor de `tickets` e colocamos de volta na fila se for maior que 0



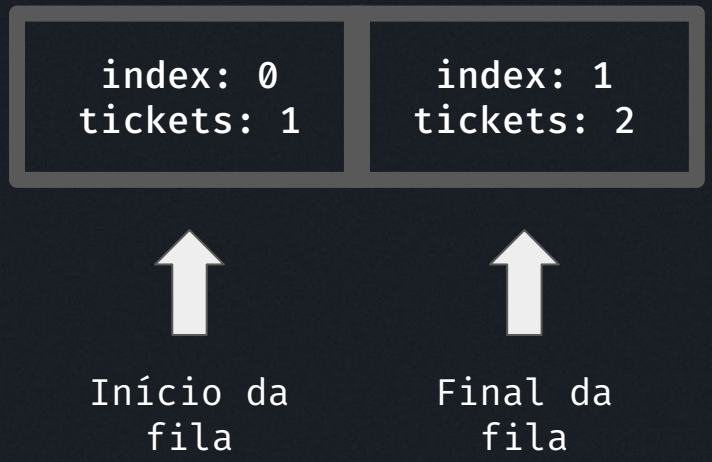
Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2
```

```
time = 3
```

Retiramos o primeiro elemento da fila e incrementamos o valor de `time`



ada

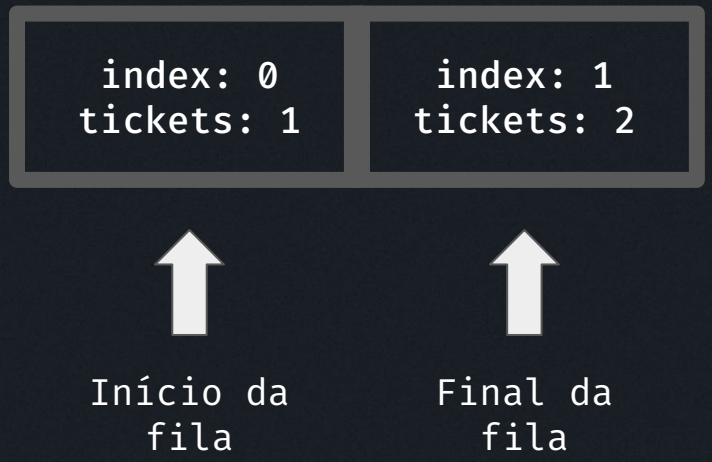
Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2
```

```
time = 3
```

Decrementamos o valor de `tickets` e colocamos de volta na fila se for maior que 0



ada

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2
```

```
time = 3
```

Decrementamos o valor de `tickets` e colocamos de volta na fila se for maior que 0

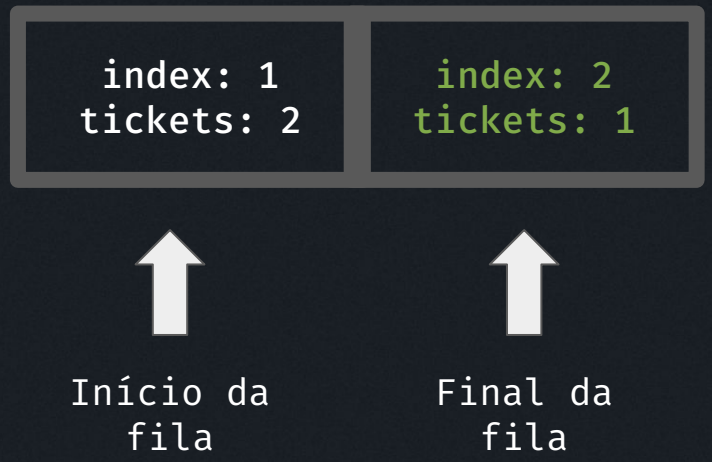
Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2
```

```
time = 4
```

Retiramos o primeiro elemento da fila e incrementamos o valor de `time`



Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2
```

```
time = 4
```

Decrementamos o valor de `tickets`. Como `tickets == 0` não colocamos de volta na fila

ada

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2


time = 4
```

Decrementamos o valor de `tickets`. Como `tickets == 0` não colocamos de volta na fila

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2
```

```
time = 5
```

Retiramos o primeiro elemento da fila e incrementamos o valor de `time`

Final da fila

Início da fila

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2


time = 5
```

Decrementamos o valor de tickets e colocamos de volta na fila se for maior que 0

Final da fila

Início da fila

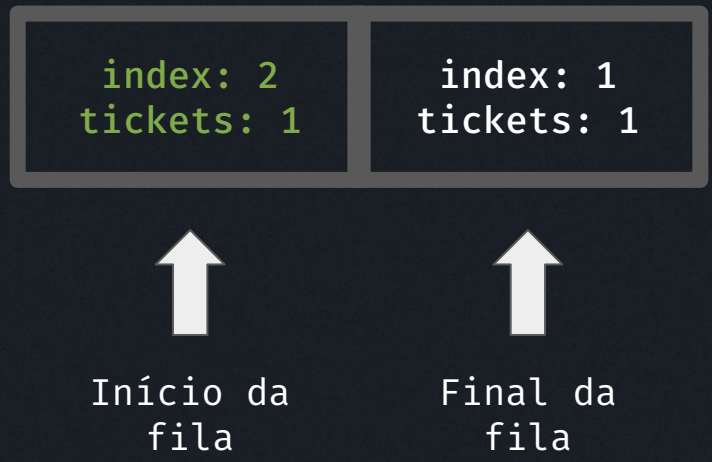
ada

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2


time = 5
```

Decrementamos o valor de `tickets` e colocamos de volta na fila se for maior que 0



Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2


time = 6
```

Retiramos o primeiro elemento da fila e incrementamos o valor de `time`

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
tickets = [2,3,2]
k = 2


time = 6
```

```
index: 2
tickets: 0
```

Decrementamos o valor de `tickets`. Agora `tickets == 0` e `index == k`. Chegamos ao fim e retornamos o valor de `time`.

```
index: 1
tickets: 1
```

Final da fila

Início da fila

Time Needed to Buy Tickets

https://leetcode.com/problems/time-needed-to-buy-tickets/

```
int TimeRequiredToBuy(int[] tickets, int k) {
    int time = 0;

    var queue = new Queue<Entry> ();
    for (int i = 0; i < tickets.Length; i++) {
        var entry = new Entry();
        entry.id = i;
        entry.tickets = tickets[i];

        queue.Enqueue(entry);
    }

    bool finished = false;
    while (!finished) {
        time++;
        var entry = queue.Dequeue();
        entry.tickets--;
        if (entry.tickets == 0 && entry.id == k) {
            finished = true;
            continue;
        }
        if (entry.tickets > 0) queue.Enqueue(entry);
    }

    return time;
}
```

```
class Entry {
    public int id;

    public int tickets;
}
```

ada

Number of Students Unable to Eat Lunch

https://leetcode.com/problems/number-of-students-unable-to-eat-lunch/

1700. Number of Students Unable to Eat Lunch

Solved

Easy | Topics | Companies | Hint

The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers `0` and `1` respectively. All students stand in a queue. Each student either prefers square or circular sandwiches.

The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a **stack**. At each step:

- If the student at the front of the queue **prefers** the sandwich on the top of the stack, they will **take it** and leave the queue.
- Otherwise, they will **leave it** and go to the queue's end.

This continues until none of the queue students want to take the top sandwich and are thus unable to eat.

You are given two integer arrays `students` and `sandwiches` where `sandwiches[i]` is the type of the i^{th} sandwich in the stack (`i = 0` is the top of the stack) and `students[j]` is the preference of the j^{th} student in the initial queue (`j = 0` is the front of the queue). Return *the number of students that are unable to eat.*

Example 1:

```
Input: students = [1,1,0,0], sandwiches = [0,1,0,1]
Output: 0
Explanation:
- Front student leaves the top sandwich and returns to the end of the line making students = [1,0,0,1].
- Front student leaves the top sandwich and returns to the end of the line making students = [0,0,1,1].
- Front student takes the top sandwich and leaves the line making students = [0,1,1] and sandwiches = [1,0,1].
- Front student leaves the top sandwich and returns to the end of the line making students = [1,1,0].
- Front student takes the top sandwich and leaves the line making students = [1,0] and sandwiches = [0,1].
- Front student leaves the top sandwich and returns to the end of the line making students = [0,1].
- Front student takes the top sandwich and leaves the line making students = [1] and sandwiches = [1].
- Front student takes the top sandwich and leaves the line making students = [] and sandwiches = [].
Hence all students are able to eat.
```

Example 2:

```
Input: students = [1,1,1,0,0,1], sandwiches = [1,0,0,0,1,1]
Output: 3
```

Constraints:

- `1 <= students.length, sandwiches.length <= 100`
- `students.length == sandwiches.length`
- `sandwiches[i]` is `0` or `1`.
- `students[i]` is `0` or `1`.

ada

Number of Students Unable to Eat Lunch

https://leetcode.com/problems/number-of-students-unable-to-eat-lunch/

Estratégia: O array `students` é uma fila (início na posição `0`) enquanto o array `sandwiches` é uma pilha (topo na posição `0`).

Podemos fazer uma simulação da execução: retira o elemento do início da fila e compara com o elemento no topo da pilha. Se forem iguais, faz um `pop` na pilha. Se forem diferentes, coloca o elemento no final da fila. Fazemos isso enquanto a fila e pilha não estiverem vazias.

Além disso, temos que contar quantas vezes tiramos um elemento da fila e colocamos de volta em sequência. Se esse número for igual ao tamanho da fila, significa que todo mundo saiu e entrou de novo na fila, ou seja, nenhum elemento da fila é igual ao topo da pilha. Nesse caso paramos o algoritmo e retornamos como resposta o número de elementos da fila.

Number of Students Unable to Eat Lunch

https://leetcode.com/problems/number-of-students-unable-to-eat-lunch/

```
public int CountStudents(int[] students, int[] sandwiches) {
    var queue = new Queue<int>(students);
    var stack = new Stack<int>(sandwiches.Reverse());

    var rotationCount = 0;
    do {
        var preference = queue.Dequeue();

        if (preference == stack.Peek()) {
            rotationCount = 0;
            stack.Pop();
        }
        else {
            rotationCount++;
            queue.Enqueue(preference);
        }

    } while (rotationCount < queue.Count && stack.Count > 0 && queue.Count > 0);

    return queue.Count;
}
```

ada

Obrigado