**INSTITUTO FEDERAL DE ALAGOAS – IFAL / CAMPUS MACEIÓ**
**Curso Bacharelado em Sistemas de Informação**
**Disciplina de Estruturas de Dados**

*Prof. MSc. Ricardo Nunes*
*Ricardo (arroba) ifal.edu.br*

# Filas (Queues)

# Objetivos

1. Entender o tipo abstrato de dados Filas
2. Ser capaz de implementar o TAD Filas usando listas em python
3. Compreender o desemepnho da implementação
4. Entender como aplicar o TAD para resolver problemas.

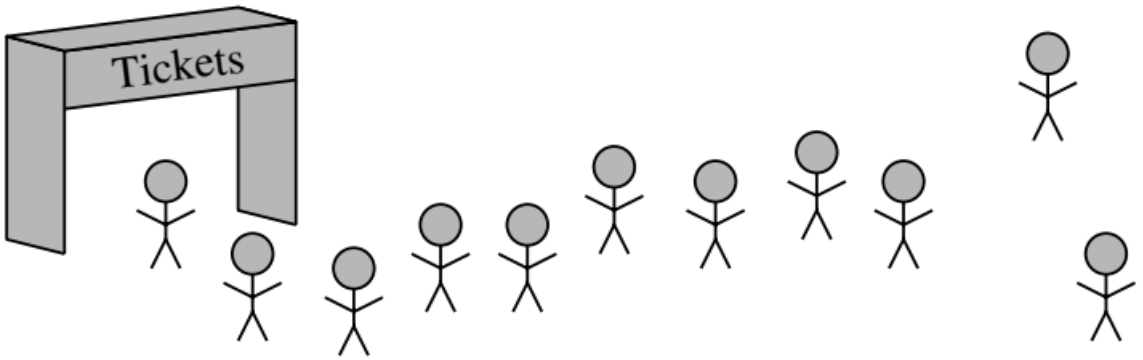
# Conteúdo

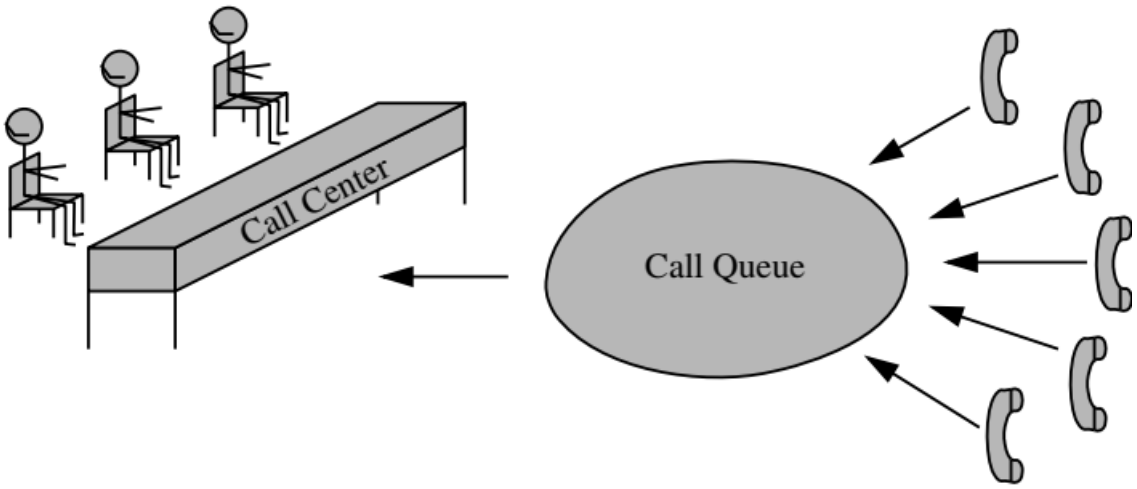
1. O que são Filas?
2. TAD Filas
3. Implementação do TAD Filas
4. Análise de desempenho
5. Aplicações

# Filas

- Coleção de objetos que são inseridos e removidos de acordo com o princípio FIFO (first in first out)
- Elementos entram na fila no final e são removidos do início

# Onde as filas são utilizadas?






# Tipo de Dado Abstrato Fila

- Coleção de objetos em sequência, onde acesso e remoção são feitos no início e inserção no final
- Uma instância do tipo Fila suporta as seguintes funções/métodos:
- F.enqueue(e): adiciona um elemento e do final da Fila F
- F.dequeue( ): remove um elemento e do início da Fila F
- Outras operações úteis:
- F.first: retorna uma referência ao primeiro elemento da fila F sem removê-lo. um erro ocorre se a fila estiver vazia
- F.is_empty: retorna True se a Fila está vazia e False caso contrário.
- len(F) ou F.size(): retorna o número de elementos na Fila.
- Está sendo assumido que não há limite de capacidade da Fila

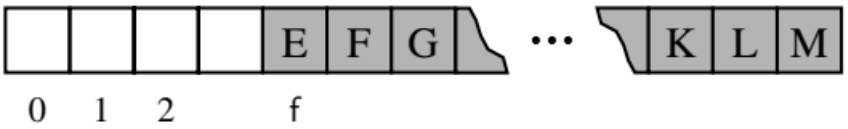
## Exemplo de série de operações com Fila

| .Valor Retornado. | .primeiro <- F <- Último. | |
|---|---|---|
| F.enqueue(5) | – | [5] |
| F.enqueue(3) | – | [5, 3] |
| len(F) | 2 | [5, 3] |
| F.dequeue() | 5 | [3] |
| F.is_empty() | False | [3] |
| F.dequeue() | 3 | [ ] |
| F.is empty() | True | [ ] |
| F.dequeue() | “error” | [ ] |
| F.enqueue(7) | – | [7] |
| F.enqueue(9) | – | [7, 9] |
| F.first() | 7 | [7, 9] |
| F.enqueue(4) | – | [7, 9, 4] |
| len(F) | 3 | [7, 9, 4] |
| F.dequeue() | 7 | [9, 4] |

# Implementação baseada em Array

- é possível adaptar a classe list do python para se comportar como uma fila usando append() e pop(0)
- o desempenho não e bom porque um loop é executado para todos os elementos para esquerda [ O(n) ]
- Qual a solução?

- colocar None na posição removida e utilizar uma variável para armazenar o índice do começo da fila

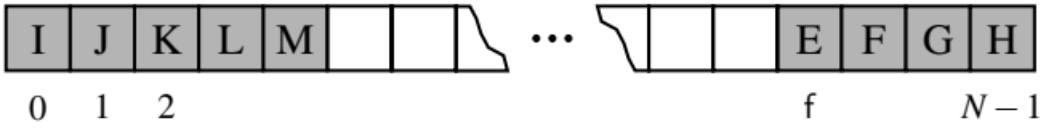



- Algum problema?

- Desperdício de espaço

# Array (Fila) Circular

- Considerando um array de tamanho fixo N, o início (I) e o final da fila são móveis…




- Novo início da fila: Ini = (Ini + 1) % N
- Fim da fila: ((Ini + Tam ) % N) - 1

# Implementação em Python

```
class FilaArray:

 CAPACIDADE_PADRAO = 5

 def __init__(self):
  self._dados = .CAPACIDADE_PADRAO
  self._tamanho = 0
  self._inicio = 0

 def __len__(self):
  return self._tamanho

 def is_empty(self):
  return self._tamanho == 0

"p">):
  if self.is_empty():
   raise FilaVazia('A Fila está vazia')
  return self._dados[self._inicio]
```

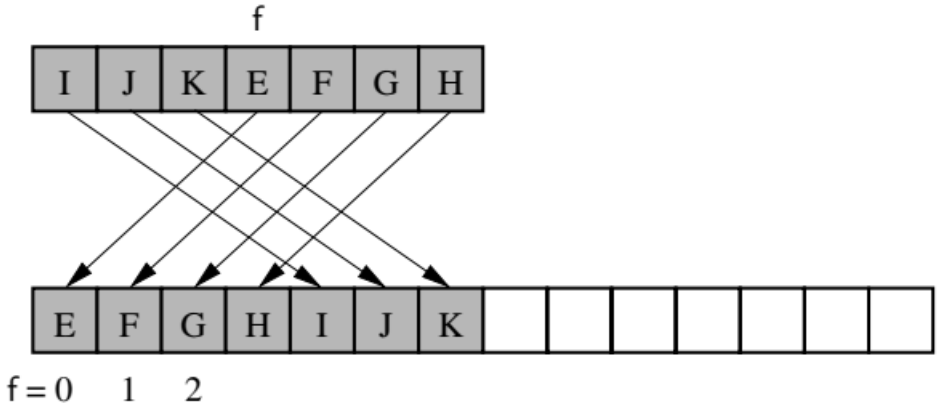
# Implementação em Python (2)

```
 def dequeue(self):
  if self.is_empty():
   raise FilaVazia('A Fila está vazia')
  result = self._dados[self._inicio]
  self._dados[self._inicio] = None
  self._inicio = (self._inicio + 1) %len(self._dados)
  self._tamanho -= 1
  return result

 def enqueue(self, e):
  if self._tamanho == len(self._dados):
   self._aumenta_tamanho(2 * len(self._dados))
  disponivel = (self._inicio + self._tamanho) %len(self._dados)
  self._dados[disponivel] = e
  self._tamanho += 1
```

# Implementação em Python (3)

```
 def _aumenta_tamanho(self, novo_tamanho):
  dados_antigos = self._dados            # keep track of existing list
  self._dados = [None] * novo_tamanho      # allocate list with new capacity
  posicao = self._inicio
  for k in range(self._tamanho):          # only consider existing elements
   self._dados[k] = dados_antigos[posicao] # intentionally shift indices
   posicao = (1 + posicao) %len(dados_antigos) # use dados_antigos size as modulus
  self._inicio = 0                    # front has been realigned
```




# Diminuindo o tamanho da Fila

- Poderia adicionar essas duas linhas em dequeue

```
if 0 < self._tamanho < len(self._dados) // 4:
  self._aumenta_tamanho(len(self._dados) // 2)
```

## Análise do desempenho da implementação da Fila

| . Operação . | . Tempo de Execução . |
|---|---|
| Q.enqueue(e) | O(1)* |
| Q.dequeue() | O(1)* |
| Q.first() | O(1) |
| Q.is_empty() | O(1) |
| len(Q) | O(1) |
| *amortized | |

# Código Fonte

https://replit.com/@RicardoRubens/filas-21

# Para estudar

- Filas
- Seções 3.10 a 3.14 do livro [5] https://panda.ime.usp.br/pythonds/static/pythonds_pt/03-EDBasicos/toctree.html
- Módulo 2, Unidade 2.3 do livro [4] http://educapes.capes.gov.br/handle/capes/176522
- Capítulo 19 (Filas) do livro [2] (texto bastante resumido!) https://chevitarese.files.wordpress.com/2009/09/aprendacomputaocompython3k.pdf

# Referências

1. Tradução do livro *Howto Think Like a Computer Scientist: Interactive Version*, de Brad Miller e David Ranum. link: https://panda.ime.usp.br/pensepy/static/pensepy/index.html
2. Allen Downey, Jeff Elkner and Chris Meyers. *Aprenda Computação com Python 3.0.* link: https://chevitarese.files.wordpress.com/2009/09/aprendacomputaocompython3k.pdf

3. SANTOS, A. C. *Algoritmo e Estrutura de Dados I*. 2014. Disponível em http://educapes.capes.gov.br/handle/capes/176522
4. SANTOS, A. C. *Algoritmo e Estrutura de Dados II*. 2014. Disponível em 2014. Disponível em https://educapes.capes.gov.br/handle/capes/176557
5. Tradução do livro [5] *Problem Solving with Algorithms and Data Structures using Python* de Brad Miller and David Ranum. link: https://panda.ime.usp.br/pythonds/static/pythonds_pt/index.html
6. Brad Miller and David Ranum. *Problem Solving with Algorithms and Data Structures using Python* link: https://panda.ime.usp.br/pythonds/static/pythonds_pt/index.html
7. Caelum. Algoritmos e Estruturas Dados em Java. Disponível em https://www.caelum.com.br/download/caelum-algoritmos-estruturas-dados-java-cs14.pdf

# That’s all Folks