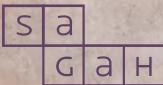


ESTRUTURA DE DADOS

Júlia Mara Colleoni Couto



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS

Implementação de filas em Python

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Identificar as regras de funcionamento de uma fila FIFO: *first in, first out*.
- Desenvolver funções e procedimentos para uma fila.
- Demonstrar as diferentes aplicações de uma fila.

Introdução

Filas são estruturas de dados dinâmicas que possibilitam resolver problemas nos quais precisamos adicionar ou remover elementos em uma lista, de forma linear. Filas são muito utilizadas em diversos sistemas na atualidade. Quando mandamos imprimir um documento, por exemplo, ele entra na fila de impressão e, se estivermos usando uma impressora compartilhada e outras pessoas estiverem imprimindo materiais ao mesmo tempo, a impressão pode demorar a sair.

Um problema computacional comum, que pode ser resolvido com o uso de filas, é o do caixeiro viajante. Imagine que ele precisa visitar determinado número de cidades e, para tanto, precisa montar uma rota, preferencialmente a mais curta, para que possa retornar para casa o mais cedo possível. Neste caso, podemos criar uma fila, na qual o caixeiro vai adicionando as cidades a serem visitadas. A primeira cidade da lista será visitada primeiro, e a última da lista será a última a ser visitada.

Neste capítulo, você vai entender o funcionamento de filas, aprendendo como os elementos são manipulados dentro dessa estrutura, a partir do uso de funções e comandos na linguagem de programação Python, versão 3. Você também verá as diversas aplicações em que as filas podem ser empregadas e aprenderá a identificar problemas que podem ser resolvidos com o uso de filas, tornando, dessa maneira, os seus projetos de *software* muito mais eficientes e dinâmicos.

1 FIFO: regras de funcionamento

Segundo Necaise (2010), uma fila é um tipo abstrato de dados, nos quais os itens podem ser inseridos apenas por um lado (retaguarda) e removidos apenas pelo outro (íncio). Ou seja, filas obedecem a ordem de chegada. Por esse motivo, filas também podem ser consideradas listas lineares, do tipo FIFO (primeiro a entrar é o primeiro a sair — do inglês *first-in, first-out*). Para fazermos consultas em filas, precisamos desenfileirar cada elemento, até que o elemento buscado seja encontrado, ou até que a fila termine. Um exemplo de fila é ilustrado na Figura 1.

De forma geral, estruturas como pilhas e filas são muito utilizadas quando ocorrem poucas inserções e remoções ao longo do tempo, e quando essas operações ocorrem apenas em posições especiais (como na frente ou no final da lista). Quando alocamos uma fila, a primeira posição é alocada no endereço 1 de memória disponível (SZWARCFITER; MARKENZON, 2010), mas podem ser utilizados ponteiros para acessar as posições desejadas. Uma fila precisa de dois ponteiros, sendo um ponteiro i para o íncio da fila e outro ponteiro r para a retaguarda. Quando uma fila estiver vazia, ambos os ponteiros apontam para o mesmo local. Isso quer dizer que, sempre que adicionamos um elemento na fila, movemos o ponteiro r , ao ponto que sempre que removemos um elemento, estaremos movendo também o ponteiro i .

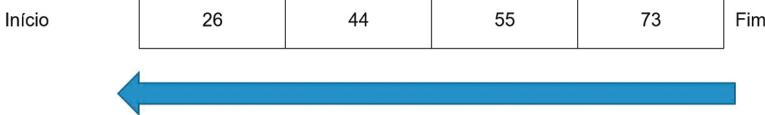


Figura 1. Visão abstrata de uma fila com quatro elementos.

Dentre as operações relacionadas a filas, Necaise (2010) aponta que existem cinco operações que são essenciais:

- `fila()`: para criar uma nova fila sem nenhum item;
- `estaVazia()`: método que deve retornar um booleano, indicando `True`, se a fila está vazia e `False`, se ela contiver algum elemento;

- `tamanho()`: deve indicar quantos itens há atualmente na fila;
 - `enfileira(item)`: adiciona um novo elemento ao final da fila;
 - `desenfileira()`: remove e retorna o primeiro elemento da fila.
- Apresenta um erro se a fila estiver vazia.

Goodrich, Tamassia e Goldwasser (2013) adicionam que o seguinte método também faz parte das operações essenciais em uma fila:

- `primeiro()`: retorna uma referência ao primeiro elemento da fila, sem removê-lo. Apresenta um erro se a fila estiver vazia.

Nesse sentido, alguns exemplos que podem ser executados, para que a fila apresentada na Figura 1 seja criada, são apresentados a seguir (Quadro 1).

Quadro 1. Operações e efeitos em uma fila **F** de números inteiros

Operação	Valor de retorno	Primeiro ← F ← Último
<code>F.fila</code>	—	[]
<code>F.estavazia()</code>	True	[]
<code>F.enfileira(34)</code>	—	[34]
<code>F.enfileira(26)</code>	—	[34, 26]
<code>F.estavazia()</code>	False	[34, 26]
<code>F.enfileira(44)</code>	—	[34, 26, 44]
<code>F.desenfileira()</code>	34	[26, 44]
<code>F.enfileira(55)</code>	—	[26, 44, 55]
<code>F.tamanho()</code>	3	[26, 44, 55]
<code>F.enfileira(73)</code>	—	[26, 44, 55, 73]
<code>F.primeiro</code>	34	[26, 44, 55, 73]

Fonte: Adaptado de Goodrich, Tamassia e Goldwasser (2013).

2 Funções e procedimentos aplicáveis em filas

Necaise (2010) explica que, em Python, há três abordagens mais comuns para a implementação de filas, visto que são tipos especializados de listas: usando um vetor (lista), uma lista encadeada ou uma matriz circular.

Implementação de filas usando vetor

A implementação mais simples é utilizando-se uma lista Python (vetor). Em listas, não temos restrição de acesso, inclusão e remoção de elementos, mas em filas temos. Por esse motivo, podemos implementar uma fila usando listas, desde que tenhamos em mente as restrições necessárias (inserção somente no fim da fila, remoção somente no início). A seguir, temos o código comentado para a implementação de uma fila usando lista Python, adaptado e baseado em Necaise (2010).

```
class Fila:
    # Cria uma fila vazia.
    def __init__(self):
        self._items = list()

    # Se a fila estiver vazia retorna True, senão retorna False.
    def estaVazia(self):
        return len(self) == 0

    # Retorna quantos elementos tem na fila.
    def __len__(self):
        return len(self._items)

    # Insere um elemento na fila.
    def enfileira(self, item):
        self._items.append(item)

    # Remove e retorna o primeiro elemento da fila.
    def desenfileira(self):
        assert not self.estaVazia(), "Fila vazia!"
        return self._items.pop(0)
```

Implementação de filas usando matriz circular

Embora a implementação de filas com listas seja mais fácil, ela também é mais custosa, pois requer tempo linear para operações de enfileiramento e desenfileiramento. Conforme Goodrich, Tamassia e Goldwasser (2013), para melhorar o desempenho dessas operações, podemos fazer a implementação de filas utilizando uma estrutura de matriz circular, que são como as matrizes lineares sendo apresentadas em formato circular. Uma das formas eficientes para se implementar uma fila é usando uma matriz circular.

Para essa implementação, precisamos manter um contador e dois apontadores, um para o início e outro para o final da matriz. O contador armazena a quantidade de itens que tem na matriz, e é importante, pois nem todos os espaços da matriz podem estar preenchidos.

Quando removemos elementos de uma matriz circular, os demais elementos não são movidos, somente o apontador de início vai apontar para o novo elemento posicionado na frente, e o contador irá decrementar um elemento. Da mesma forma, quando inserirmos elementos, o apontador do final da fila irá apontar para o novo elemento, e o contador irá incrementar um elemento. Na Figura 2, temos uma representação circular e outra linear da mesma matriz, mostrando os apontadores de início e fim, e o contador, que indica que temos seis elementos na matriz.

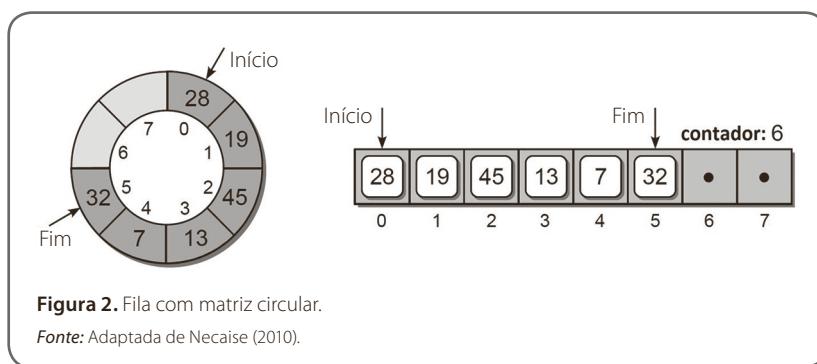


Figura 2. Fila com matriz circular.

Fonte: Adaptada de Necaise (2010).

Agora, vamos mostrar como seria a implementação com uma matriz circular em Python adaptada e baseada em Goodrich, Tamassia e Goldwasser (2013):

```
class FilaMatrizCircular:
    # Cria uma fila vazia, utilizando a estrutura de uma matriz
    # circular
    CAPACIDADE = 10 # limite de capacidade das filas novas

    def __init__(self):
        self._data = [None] * FilaMatrizCircular.CAPACIDADE
        self._tamanho = 0
        self._frente = 0

    # Retorna quantos elementos tem na fila.
    def __len__(self):
        return self._tamanho

    # Se a fila estiver vazia retorna True, senão retorna False.
    def estaVazia(self):
        return self._tamanho == 0

    # Retorna o primeiro elemento da fila, sem removê-lo
    def primeiro(self):
        if self.estaVazia():
            raise Empty("Fila vazia!")
        return self._data[self._frente]

    # Insere um elemento na fila.
    def enfileira(self, e):
        if self._tamanho == len(self._data):
            self._redimensiona(2 * len(self._data)) # duplica o tamanho
        avail = (self._frente + self._tamanho) % len(self._data)
        self._data[avail] = e
        self._tamanho += 1

    # Remove e retorna o primeiro elemento da fila.
    def desenfileira(self):
        if self.estaVazia():
            raise Empty("Fila vazia!")
        resposta = self._data[self._frente]
        self._data[self._frente] = None # help garbage collection
        self._frente = (self._frente + 1) % len(self._data)
        self._tamanho -= 1
        return resposta

    # Redimensiona a lista, cap >= len(self)
    def _redimensiona(self, cap):
        antiga = self._data # mantém o registro pra lista existente
        self._data = [None] * cap # aloca uma lista com nova capacida
de
```

```
anda = self._frente
for k in range(self._tamanho): # com base nos elementos existentes
    self._data[k] = antiga[anda] # altera os indices
    anda = (1 + anda) % len(antiga) # usa o tamanho antigo como módulo
self._frente = 0 # realinha a frente
```



Fique atento

Ao utilizarmos uma estrutura de matriz circular, sempre que o marcador atingir o último elemento da matriz linear real, ele deverá devolver o primeiro elemento da matriz.

Implementação de filas usando listas encadeadas

Podemos também implementar filas utilizando listas simplesmente encadeadas. Em listas desse tipo, cada elemento (ou nó) armazena um ponteiro para o próximo elemento da lista. Neste caso, como inserimos em uma ponte e o removemos em outra, teremos que guardar dois ponteiros, um para o primeiro e outro para o último elemento, conforme ilustrado na Figura 3. Neste caso, ao remover um elemento, precisamos que o ponteiro para o início aponte para o sucessor do elemento removido, assim como, ao inserir um novo elemento, precisamos que o ponteiro para o fim aponte para esse novo elemento que foi inserido.

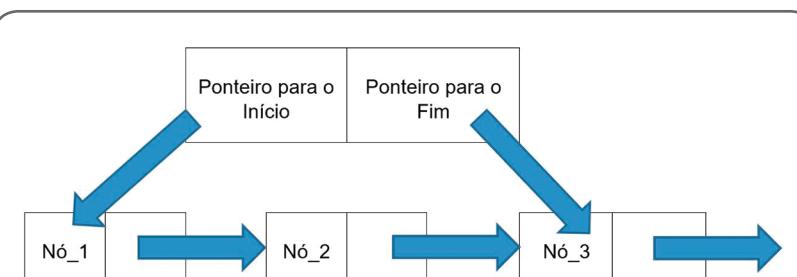


Figura 3. Fila com lista encadeada simples.

Aqui, vamos mostrar uma possível implementação em Python usando uma lista encadeada, baseada em Necaise (2010).

```
class FilaListaEncadeada :  
    # Cria uma fila vazia, utilizando a estrutura de uma lista encadeada.  
    def __init__(self):  
        self._inicio = None  
        self._fim = None  
        self._contador = 0  
  
    # Se a fila estiver vazia retorna True, senão retorna False.  
    def estaVazia(self):  
        return self._inicio is None  
  
    # Retorna quantos elementos tem na fila.  
    def __len__(self):  
        return self._contador  
  
    # Insere um elemento na fila.  
    def enfileira(self, item):  
        elemento = _NoFila(item)  
        if self.estavazia() :  
            self._inicio = elemento  
        else :  
            self._fim.next = elemento  
        self._fim = elemento  
        self._contador += 1  
  
    # Remove e retorna o primeiro elemento da fila.  
    def desenfileira(self):  
        assert not self.estavazia(), "Fila vazia!"  
        elemento = self._inicio  
        if self._inicio is self._fim :  
            self._fim = None  
  
        self._inicio = self._inicio.next  
        self._contador -= 1  
        return elemento.item  
  
    # Classe privada para armazenamento de nós da lista encadeada.  
class _NoFila(object):  
    def __init__(self, item):  
        self.item = item  
        self.next = None
```

A diferença entre as três implementações, além da complexidade do código, é no desempenho das operações. A implementação de matriz circular é mais eficiente se comparada com listas. Em uma matriz circular, todas as operações têm a pior complexidade de tempo de $O(1)$, pois não precisamos mudar os itens da matriz de lugar. Contudo, uma das desvantagens da matriz circular é a de limitar o tamanho disponível para a alocação de elementos. Finalmente, a implementação com listas encadeadas permite operações de inserção mais rápidas que a matriz circular, pois não necessitam percorrer do início até o fim da fila para poder executar a inserção, contudo, o problema é o desenfileiramento, que requer tempo $O(n)$, pois toda a lista deve ser pesquisada para encontrar o item a ser removido.

Implementação de filas usando o módulo Python `queue`

Python tem um módulo chamado `queue`, que implementa uma fila com multiprodutores e multiconsumidores. Os métodos públicos implementados com essa biblioteca são os seguintes:

- `Queue.qsize()`: tamanho da fila.
- `Queue.empty()`: verifica se a fila está vazia (retorna `True`, se estiver vazia, e `False`, se não).
- `Queue.full()`: verifica se a fila está cheia (retorna `True`, se estiver cheia, e `False`, se não).
- `Queue.put(item)`: adiciona item na fila.
- `Queue.put_nowait(item)`: adiciona item na fila, sem bloqueio.
- `Queue.get()`: remove e retorna um item da fila.
- `Queue.get_nowait()`: remove e retorna o primeiro item da fila, sem bloqueio.

Com este módulo, além de filas do tipo FIFO, também é possível implementar estruturas que seguem a abordagem LIFO (p. ex., pilhas, ou seja, último a entrar, primeiro a sair) e filas prioritárias.



Link

Confira mais informações sobre o módulo `queue` no *link* a seguir.

<https://qrgo.page.link/ZZ4sf>

3 Aplicações de uma fila

A seguir, vamos mostrar exemplos utilizando os tipos de filas que criamos na seção anterior. Usando como base a primeira estrutura de fila criada, podemos fazer algumas operações, conforme o seguinte exemplo:

```
f = Fila()
print(list(f._items))      # []
print(f.estaVazia())      # True
f.enfileira(34)
print(list(f._items))      # [34]
f.enfileira(26)
print(list(f._items))      # [34, 26]
f.estaVazia()
f.enfileira(44)
print(list(f._items))      # [34, 26, 44]
f.desenfileira()
print(list(f._items))      # [26, 44]
f.enfileira(55)
print(list(f._items))      # [26, 44, 55]
print(f.__len__())         # 3
f.enfileira(73)
print(list(f._items))      # [26, 44, 55, 73]
```

Para alterar os exemplos e utilizar as filas com matriz circular, bastaria adaptar:

```
f = FilaMatrizCircular()
print(f._data)          # [None, None, None, None, None, None, None
, None, None]
print(f.estaVazia())    # True
f.enfileira(34)
print(f._data)          # [34, None, None, None, None, None, None,
None, None]
f.enfileira(26)
print(f._data)          # [34, 26, None, None, None, None, None, No
ne, None, None]
f.estaVazia()
f.enfileira(44)
print(f._data)          # [34, 26, 44, None, None, None, None, None
, None, None]
f.desenfileira()
print(f._data)          # [None, 26, 44, None, None, None, None, No
ne, None, None]
f.enfileira(55)
print(f._data)          # [None, 26, 44, 55, None, None, None, None
, None, None]
print(f.__len__())      # 3
f.enfileira(73)
print(f._data)          # [None, 26, 44, 55, 73, None, None, None,
None, None]
```

Para criar uma lista com uma fila encadeada, você pode proceder da mesma forma, adaptando a criação e os métodos de impressão da fila. Também podemos utilizar o módulo Python `queue`. Assim sendo, para criarmos uma fila com a estrutura `[12, 40, 33, 15, 21, 100, 17]`, teríamos a seguinte sequência de comandos em Python:

```
import queue
fila = queue.Queue()
fila.put(12)
fila.put(40)
fila.put(33)
fila.put(15)
fila.put(21)
fila.put(100)
fila.put(17)
print(list(fila.queue))
# Resultado: [12, 40, 33, 15, 21, 100, 17]
```

Depois de criar um objeto de fila, podemos fazer as operações sobre ela, e após o sinal de #, podemos ver o resultado que teremos no console:

```
print(list(fila.queue)) # [12, 40, 33, 15, 21, 100, 17]
print(fila.qsize())      # 7
print(fila.empty())     # False
print(fila.full())      # False
fila.put(5)              # adiciona elemento 5 na fila
print(list(fila.queue)) #[12, 40, 33, 15, 21, 100, 17, 5]
fila.put_nowait(55)      #adiciona elemento 55 na fila
print(list(fila.queue)) #[12, 40, 33, 15, 21, 100, 17, 5, 55]
fila.get(15)              # remove e retorna um item da fila
print(list(fila.queue)) #[40, 33, 15, 21, 100, 17, 5, 55]
fila.get_nowait()        # remove o primeiro elemento, sem bloqueio.
print(list(fila.queue)) #[33, 15, 21, 100, 17, 5, 55]
```



Exemplo

Em um aeroporto, precisamos sempre contar com duas filas — uma para o pouso e outra para a decolagem dos aviões. Em aeroportos menores, como a pista de decolagem e de pouso é única, a fila de espera, tanto dos aviões que vão pouso quanto dos que vão decolar, é a mesma, sendo que cada piloto deve esperar a sua vez e a autorização para poder prosseguir.

As estruturas de dados do tipo fila podem ser empregadas em diversas situações práticas. Algumas situações são mapeadas por Goodrich, Tamassia e Goldwasser (2013), como, por exemplo, quando ligamos para um *call center* e todas as posições de atendentes estão ocupadas, sendo assim, a nossa ligação entra em uma fila, na qual o primeiro dela será o primeiro a ser atendido quando um dos atendentes liberar a linha telefônica. *Chats on-line* com operadoras de telefonia ou bancos também funcionam da mesma forma, em que precisamos aguardar algum atendente se liberar para que possa nos atender.

Outro exemplo seria quando entramos em uma fila para comprar ingressos para um *show musical*. Essas filas também ocorrem *on-line*, sendo assim, precisamos aguardar a nossa vez para podermos concretizar a compra. Adicionalmente, filas podem ser utilizadas para organizar o atendimento de processos requisitados ao sistema operacional e para ordenar o encaminhamento de pacotes em um roteador de rede.



Referências

GOODRICH, M. T.; TAMASSIA, R.; GOLDWASSER, M. H. *Data structures and algorithms in Python*. Hoboken: John Wiley & Sons, 2013.

NECAISE, R. D. *Data structures and algorithms using Python*. Hoboken: John Wiley & Sons, 2010.

SZWARCFITER, J. L.; MARKENZON, L. *Estruturas de dados e seus algoritmos*. 3. ed. São Paulo: LTC, 2010.



Fique atento

Os *links* para sites da web fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declararam não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS