CENTRO DE CIÊNCIA E TECNOLOGIA
LABORATÓRIO DE CIÊNCIAS MATEMÁTICAS
UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE

TAD Fila
Implementações em Python

Disciplina: Estrutura de Dados I

Prof. Fermín Alfredo Tang Montané

Curso: Ciência da Computação

TAD Fila (Queue ADT)
Usando uma lista Python

- Uma fila é uma estrutura de dados que é uma coleção linear de elementos cujo acesso está restrito a regra FIFO. Novos itens são inseridos no fim da fila, enquanto elementos presentes na fila são removidos pela frente. Os itens são mantidos na ordem em que foram adicionados a estrutura.
- As principais operações são:
  - Queue(): Cria uma fila vazia;
  - isEmpty(): Retorna um valor booleano indicando se a fila está vazia;
  - length(): Retorna o número de elementos presentes na fila;
  - enqueue(item): adiciona o elemento item no final da fila;
  - dequeue(): remove e retorna o elemento que está na frente da fila.
- Apresenta-se três formas de implementar uma fila em Python:
  - Usando uma lista em Python;
  - Usando array circular;
  - Usando listas encadeadas.

TAD Fila (Queue ADT)
Usando uma lista Python

- A maneira mais simples de implementar uma fila em Python é usando uma lista Python. A lista Python pode ser considerada um caso mais geral de fila. A estrutura já possui funções para adicionar e remover elementos.
- No entanto, devemos lembrar que toda lista Python opera sobre um array e as operações realizam deslocamentos de elementos, redimensiomento do array, que implica na copia de dados em um novo array, o que pode afetar o desempenho.
- A figura ilustra uma fila como lista Python.

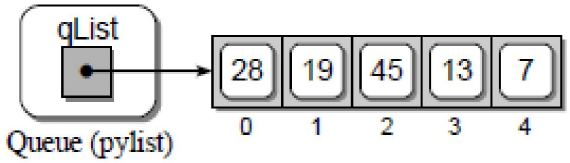


Figure 8.3: An instance of the Queue ADT implemented using a Python list.

TAD Fila (Queue ADT)
Usando uma lista Python

- A figura ilustra as operações de remoção (dequeue) e inserção (enqueue) em uma lista python.

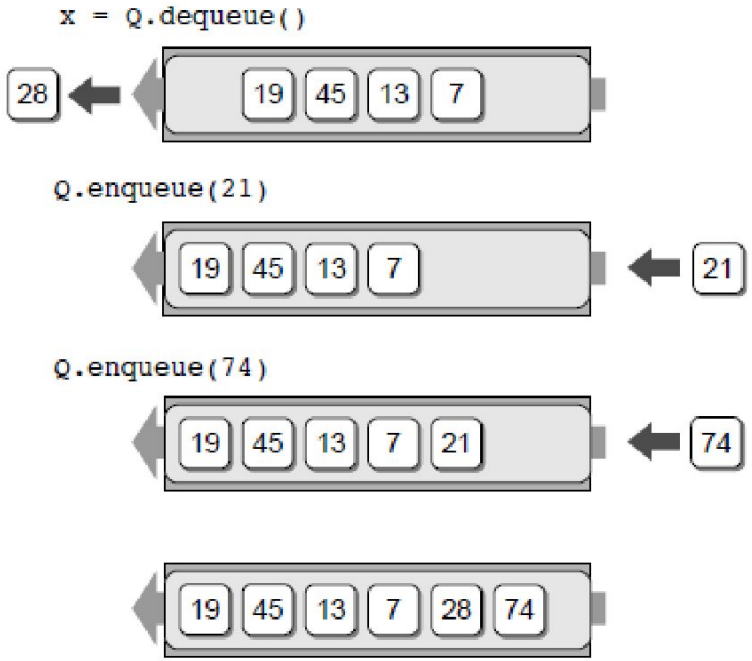


Figure 8.2: Abstract view of the queue after performing additional operations.

TAD Fila (Queue ADT)
Usando uma lista Python

- A implementação de uma fila usando uma lista Python é bastante simples e direta, usando os métodos append e pop disponíveis em qualquer lista Python.

- O módulo pylistqueue.py que implementa o TAD Fila (Queue ADT) possui uma a classe Queue.

Listing 8.1 The pylistqueue.py module

```
1  # Implementation of the Queue ADT using a Python list.
2  class Queue :
3     # Creates an empty queue.
4    def __init__( self ):
5      self._qList = list()
6
7     # Returns True if the queue is empty.
8    def isEmpty( self ):
9      return len( self ) == 0
10
11    # Returns the number of items in the queue.
12   def __len__( self ):
13     return len( self._qList )
14
15    # Adds the given item to the queue.
16   def enqueue( self, item ):
17     self._qList.append( item )
18
19    # Removes and returns the first item in the queue.
20   def dequeue( self ):
21     assert not self.isEmpty(), "Cannot dequeue from an empty queue."
22     return self._qList.pop( 0 )
```

Classe Queue

Construtor da Fila

Fila Vazia

Comprimento da Fila

Insere item na Fila

Remove da Fila

TAD Fila (Queue ADT)
Usando Array Circular

- A implementação baseada em listas em Python é fácil de implementar porém requer tempo linear $O(n)$ nas operações de inserção e remoção, devido a que em toda remoção ou inserção pode ser necessário compactar ou expandir o array associado. Em particular no caso da remoção de um elemento é necessário deslocar os elementos restantes para frente.
- Um array circular consiste na ideia de enxergar um array como um circulo em vez de uma linha. Considera-se que, imediatamente após a ultima posição do array segue a primeira posição. Ver Figura.

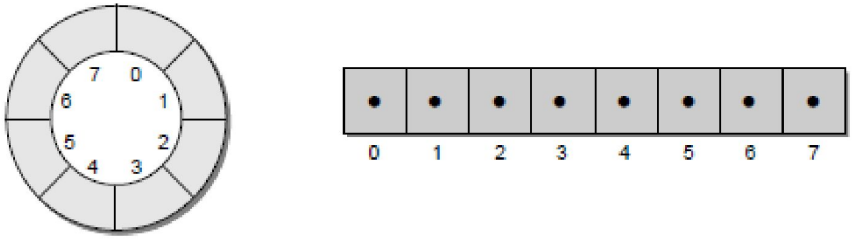


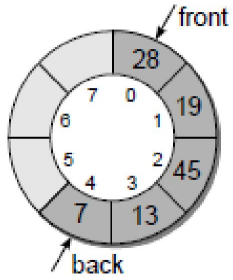
Figure 8.4: The abstract view of a circular array (left) and the physical view (right).

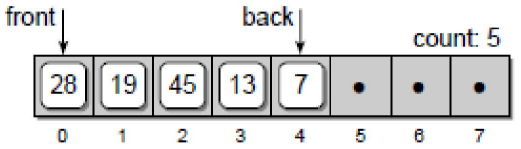
- O array circular elimina a necessidade de deslocamento de elementos, mas introduz o limite na capacidade máxima da fila.

TAD Fila (Queue ADT)
Usando Array Circular

- Para implementar uma fila mediante um array circular, precisamos de um **contador** (count) para o número de elementos, e **dois índices** para marcar as posições de inicio (front) e de fim (back) da fila.
- Observe como estes campos mudam no exemplo.

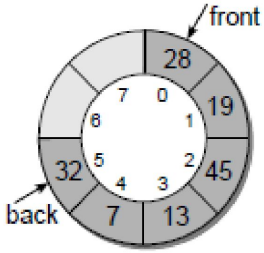
(a)

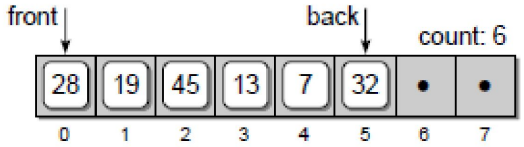




enqueue (32)

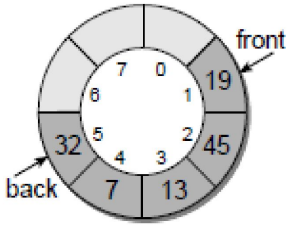
(b)

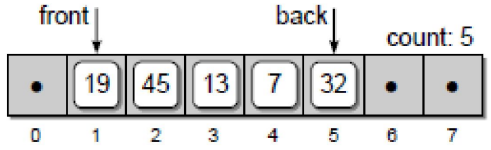




dequeue

(c)

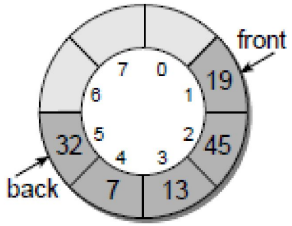


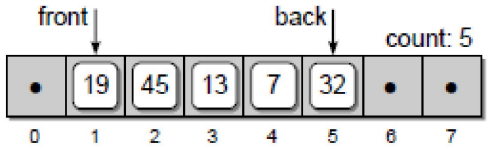


TAD Fila (Queue ADT)
Usando Array Circular

- Neste exemplos, observe que os índices front e back mudam de posição.
- No exemplo (d), observe que back atinge a última posição.
- No exemplo (e), uma nova inserção faz com que a próxima posição de back seja 0.

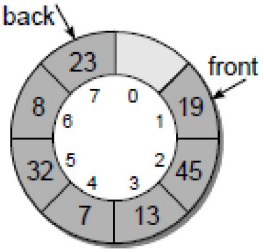
(c)

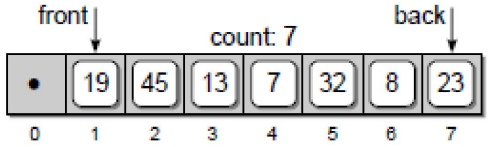




enqueue (8)
enqueue (23)

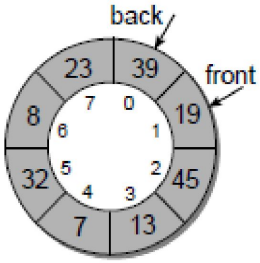
(d)

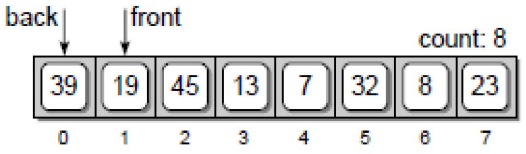




enqueue (39)

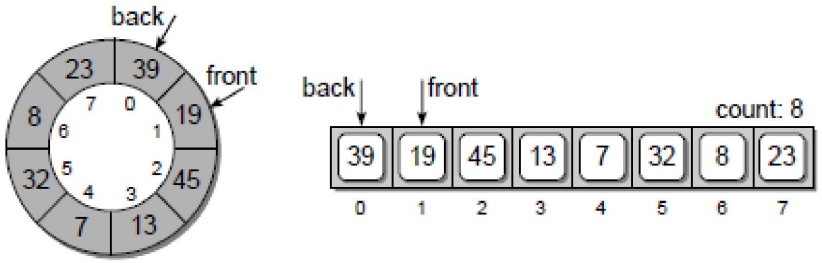
(e)





TAD Fila (Queue ADT)
Usando Array Circular

- No último exemplo, com fila cheia, back precede a front.



- Define-se condição semelhante para fila vazia, back precede a front.

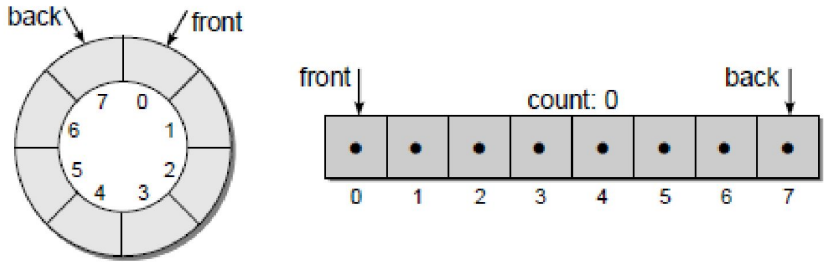


Figure 8.6: The circular array when the queue is first created in the constructor.

TAD Fila (Queue ADT)
Usando Array Circular

- A implementação usando um array circular é ilustrada na figura.
- Esta representação utiliza uma estrutura cabeçalho contendo o tamanho da fila (count), o índice do primeiro elemento da fila (front), o índice do último elemento da fila (back), e um ponteiro ao array que representa a fila (qArray).
- Devido a que a estrutura de array possui um tamanho fixo é necessário utilizar uma função isFull() para verificar se a fila está cheia.

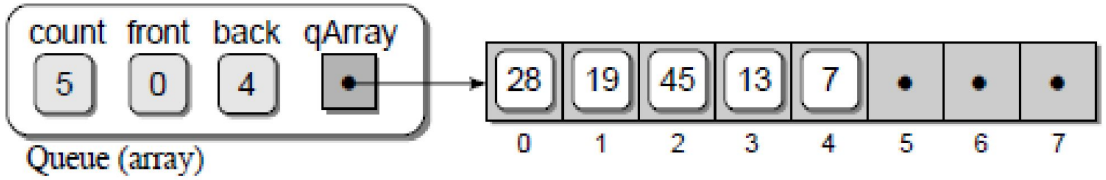


Figure 8.5: A Queue object implemented as a circular array.

- Neste caso as operações de inserção e remoção requerem tempo constante $O(1)$.

TAD Fila (Queue ADT)
Usando Array Circular

- O módulo arrayqueue.py que implementa o TAD Fila (Queue ADT) importa a classe Array do módulo array.py
- Ela possui uma única classe:
  - A classe Queue que implementa a fila e é pública.

Listing 8.2 The arrayqueue.py module.

```
1  # Implementation of the Queue ADT using a
2  from array import Array
3
4  class Queue :
5      # Creates an empty queue.
6    def __init__( self, maxSize ) :
7      self._count = 0
8      self._front = 0
9      self._back = maxSize - 1
10     self._qArray = Array( maxSize )
11
12     # Returns True if the queue is empty.
13   def isEmpty( self ) :
14     return self._count == 0
15
16     # Returns True if the queue is full.
17   def isFull( self ) :
18     return self._count == len(self._qArray)
19
20     # Returns the number of items in the queue.
21   def __len__( self ) :
22     return self._count
```

Classe Queue

Construtor da Fila

Inicializa o contador, os índices e cria o Array

Fila Vazia

Fila Cheia

Comprimento da Fila

TAD Fila (Queue ADT)
Usando Array Circular

- Em ambas funções, enqueue() e dequeue(), o calculo mais importante é a atualização da posição de inserção (back) e de remoção (front), respectivamente, avançando uma posição ou dando a volta de forma circular. Pode ser feito como em cada caso, como:

```
self._back += 1
if self._back == len( self._qArray ) :
  self._back = 0
```

```
self._front +=1
if self._front == len( self._qArray ) :
   self._front  = 0
```

- Uma forma mais eficiente é usando o operador módulo como mostrado no código.

```
23
24      # Adds the given item to the queue.
25      def enqueue( self, item ):
26        assert not self.isFull(), "Cannot enqueue to a full queue."
27        maxSize = len(self._qArray)
28        self._back = (self._back + 1) % maxSize
29        self._qArray[self._back] = item
30        self._count += 1
31
32      # Removes and returns the first item in the queue.
33      def dequeue( self ):
34        assert not self.isEmpty(), "Cannot dequeue from an empty queue."
35        item = self._qArray[ self._front ]
36        maxSize = len(self._qArray)
37        self._front = (self._front + 1) % maxSize
38        self._count -= 1
39        return item
```

Insere item na Fila

Atualiza a posição back

Remove da Fila

Atualiza a posição front

TAD Fila (Queue ADT)
Usando Lista Encadeada

- A implementação usando lista encadeada é ilustrada na figura.
- Esta representação utiliza uma estrutura cabeçalho contendo o tamanho da fila (size ou count), um ponteiro ao último elemento da fila (qtail) e um ponteiro ao primeiro elemento da fila (qhead).

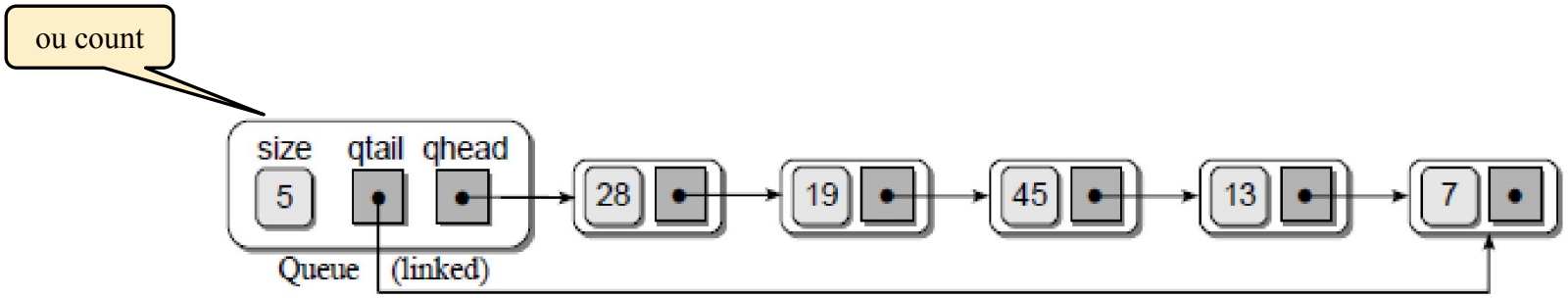


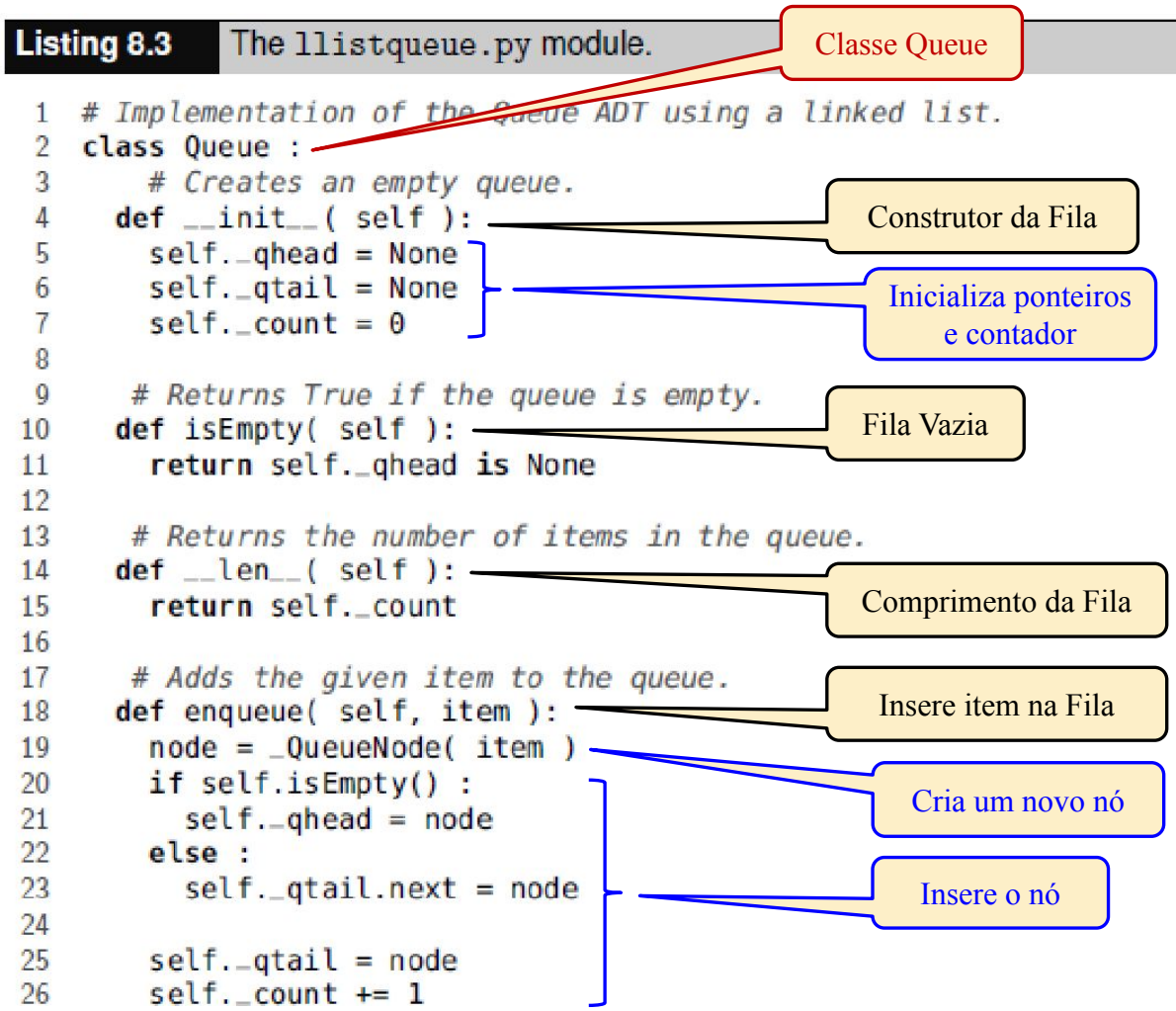
Figure 8.7: An instance of the Queue ADT implemented as a linked list.

TAD Fila (Queue ADT)
Usando Lista Encadeada

- O módulo llistqueue.py que implementa o TAD Fila (Queue ADT) possui duas classes:
  - A classe Queue que implementa a fila e é pública.
  - A classe _QueueNode que implementa um nó da lista encadeada e é privada.

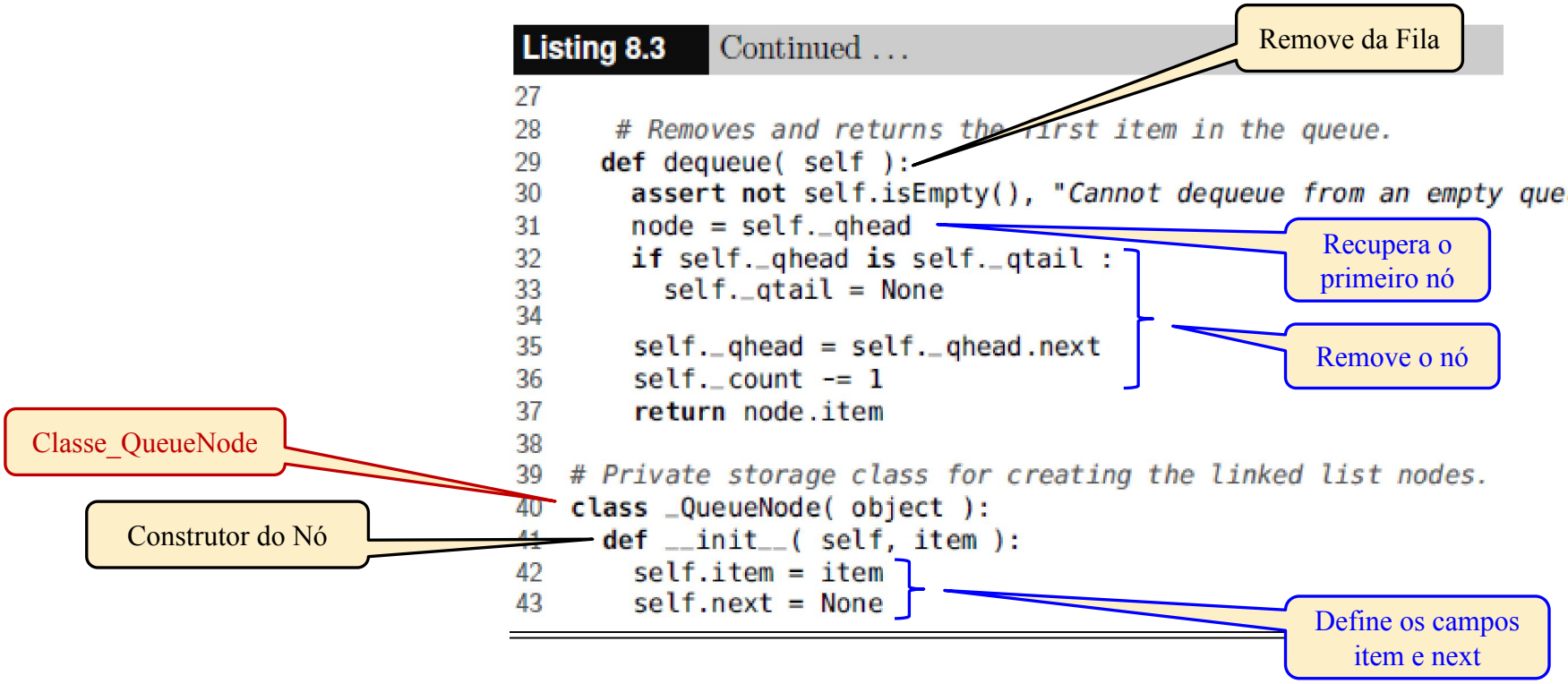
Listing 8.3 The `llistqueue.py` module.

```
1  # Implementation of the Queue ADT using a linked list.
2  class Queue :
3      # Creates an empty queue.
4    def __init__( self ):
5      self._qhead = None
6      self._qtail = None
7      self._count = 0
8
9     # Returns True if the queue is empty.
10   def isEmpty( self ):
11     return self._qhead is None
12
13    # Returns the number of items in the queue.
14   def __len__( self ):
15     return self._count
16
17    # Adds the given item to the queue.
18   def enqueue( self, item ):
19     node = _QueueNode( item )
20     if self.isEmpty() :
21       self._qhead = node
22     else :
23       self._qtail.next = node
24
25     self._qtail = node
26     self._count += 1
```



TAD Fila (Queue ADT)
Usando Lista Encadeada

- Usando lista encadeada.



Referências

- Rance Necaise. Data Structures and Algorithms Using Python. Capítulo 8. Queues. 2011.